Chapter 1: PACS Photometry standard data processing

1 Introduction

This chapter describes the standard processing steps for the different photometry observation modes of the PACS instrument. For every step it gives a brief description of the algorithms (optimizations of complexity are beyond the scope of this document) and calibration tables that are needed as input. The different intermediate conceptual formats of the PACS photometry data throughout the reduction are described as well.

2 Important Note

This documentation refers to the latest available version of the PACS photometer pipeline. The version used during this DP Workshop is not the latest version. Therefore, this documentation does not reflect the status of the pipeline as seen by the users during the workshop. We list below all the known bugs and differences of the pipeline DP Workshop version with respect to this documentation

1. the calblock pre-processing tasks are not available in the DP Workshop version; those tasks can not be used
2. the photMMTDeglitching task is not producing the MMT_GLITCH mask: this is solved in the latest pipeline version.
3. the photDiffChop task of the point and small source pipeline is not working properly; it might crash due to a known and now solved bug
4. the astrometry is still not accurate; this is due to a wrong treatment of the Position Angle in the photAddInstantPointing and photAssignRaDec tasks
5. the photDriftCorrection task is not working due to the missing input (output of cal-block pre-processing tasks); thus, the data can only be partially flux calibrated by the photRespFlatFieldCorrection task.
6. the photProject task is not flux conserving when the output pixel size is different from the input pixel size; this is due to a well known and now solved bug. The astrometry of the output map is not accurate due to a problem (now solved) with the wcs.
7. the exposure, coverage and noise maps are not properly propagated through the pipeline steps; the proper treatment is under development
8. the use of the masks is still not accurate and the propagation of the masks in the pipeline steps is not appropriate; the proper mask treatment is under development
9. the level 2 product of the Point-source pipeline (final image with astrometric calibration) is not yet available in the DP workshop pipeline version. A prototype is available and under testing in the latest version.

3 Summary of the Photometry processing steps

We summarize here the basic steps of the PACS photometry data reduction. The aim of this chapter is to explain the user how to reduce the PACS photometry data starting by different "Level Product". We assume here that the user is familiar with the concept of the "ObservationContext". So we assume that the user will start the data processing by accessing different levels of data Products in her/his local store. Under these assumption the basic steps of the data processing starting from Level 0 Products are the following:

1. access the local store and retrieve the Frames of a given observation and the related pointing product
2. identify the structure of the observation and identify the main block (Calibration and Science blocks)

3. pre-process the calibration block and extract useful information for the further calibrations
4. perform data cosmetics: flag bad/saturated pixels and flag/correct cross talk and glitches
5. convert signal from digits to volts
6. covert chopper position from engineering units into angle
7. satellite pointing info are added to frames (sky coordinates of reference pixel for each readout)
8. the astrometry is calculated on the basis of spatial calibration files (spatial distortions are taken into account)
9. in case of chopped observation the chop-nod cycle is reduced to remove sky and telescope background
10. the flat field and flux calibration are applied and corrected for possible drifts
11. The spacecraft on-board time is converted to UTC
12. in case of scan map observation, the signal is filtered to remove 1/f noise
13. A stack/mosaic of frames is constructed

4 Processing levels

There is a Herschel-wide convention on processing levels of the different instruments. Here we list the content and the properties of the different Product Level for the PACS Photometry mode.

- *Raw Telemetry :* This is the format of the raw PACS photometry data. The telemetry file is composed of telemetry packets produced by the instrument in the course of the observation. These data are pre-processed and compressed on board of Herschel. For pre-processing we mean a simple averaging any 4 readouts for a final sampling of 10 Hz. This data product will not be visible in the pipeline processing and it will not be delivered to the end user.
- *Decompressed Science Data :* This is an "artificial level". The data are not stored and not visible for general user. But in the interactive step by step data analysis the data product can be analyzed for debugging purposes. Telemetry data as measured by the instrument, minimally manipulated and stored as Data Frames. For PACS photometry, this level is stored/manipulated in a *DataFrameSequence* : a sequence of PACS dataframes, which are decompressed SPU buffers. What is contained in every decompressed SPU buffer depends on the SPU reduction mode. Typically there are several reduced readouts for every active detector (averaged detector signals), 40Hz or 20Hz readouts for a few selected pixels and mechanism/status information sampled at 40Hz/20Hz by the DecMec, the so-called DMC Header.
- *Level 0 data:* Level 0 data is a complete set of data requested to do the scientific data reduction. It is saved in a Level 0 Data Pool in form of Fits files. After Level 0 data generation no connection to the Database is possible any more. Therefore Level 0 data need to contain all information needed from the Database (e.g. uplink information).
  - Science data Science data are organized in user friendly classes. The`Frames` class for reduced data and the`PhotRaw` class for additional raw channel data will be the basic data products for this processing steps. The so-called Status table of the Frames class stores the info carried by the DMC header which are necessary for the data reduction (chopper position, identification of internal calibration observation and scientific observations).
  - Auxiliary data Auxiliary data for the time span covered by the Level 0 data, such as the spacecraft pointing (attitude history), the time correlation, selected spacecraft housekeeping, etc The information is partly merged as status entries into the basic science classes Frames and PhotRaw or available as Products (Pointing)
  - Decoded HK Data HK data Tables with converted and raw HK values.
  - Calibration files and data of 'associated' observations - e.g. photometric checks or other Trend Analysis results taken throughout the operational day or even before (still to be clarified!).
- *Level 0.5 data :* Processing until Level 0.5 is AOT independent These data are saved in the Product Pool. On this Level additional information is added to the Frames class (Flags for Saturation, Flags Bad Pixel, BlockTable,...) and basic unit conversion are done (digital values to Volts, chopper angle).

- *Level 1 data:* Level 1 data generation is AOT dependent. Level 1 data are saved in the Product Pool. Detector readouts calibrated, converted to physical units and grouped into blocks. For PACS photometry this is a data cube with flux densities with associated sky coordinates. Mostly every step before actual Image construction is done. The `Frames or FramesStack` class will be the basic Level 1 product of photometer data Possibly the Level 1 data generation can be done automatically to a large extend after the instrument has been calibrated.
- *Level 2 data:* Further processed level-1 data to such a level that scientific analysis can be performed. The noise is filtered and the map is reconstructed with different methods/algorithms depending on the AOT mode. For optimal results many of the processing steps involved to generate level-2 data may require human interaction, based both on instrument understanding as well as understanding of the scientific aims of the observation. The result is an Image Product.
- *Level 3 data:* These are the publishable science products where level-2 data products are used as input. These products are not only from the specific instrument, but are usually combined with theoretical models, other observations, laboratory data, cataloguers, etc. Their formats should be VO compatible and these data products should be suitable for VO access.

5 Imports

To be able to execute the commands in this document, you need to import the necessary java classes and jython toolboxes:

```
>> from all import *
```

6 Used Masks

The following Masks are used by default during Pipeline processing, additional masks can be added by the user when necessary:

```
BLINDPIXEL  : Pixel masked out by the DetectorSelectionTable (already in Level 0)
BADPIXEL    : Bad Pixel masked during pipeline processing
SATURATION  : Saturated Readouts
MMTGLITCH      : Glitched Readouts
UNCLEANCHOP : Flagging unreliable signals at the begin and end of a ChopperPlateau
```

All the masks created by the pipeline are 3D masks. This is true even for the cases when it is not necessary such as in the BLINDPIXEL, BADPIXEL and UNCLEANCHOP masks. Moreover all the masks are boolean: unmasked pixels are saved as FALSE and masked pixels are saved as TRUE. Be careful that the DatasetInspector in jide and the Editor in hipe are not able to properly transform a boolean mask into integer mask. Due to this bug, the table shown by the jide DatasetInspector and the hipe Editor for each mask has wrong dimension. Only the MaskViewer is able to properly display the masks (see section 10.6.6.2 of the PACS User Manual for details about the use of the MaskViewer). Few words should de spent about the BLINDPIXEL mask. This is an uplink mask, used during commanding, to indicate which pixels should not be read at all. Currently, the BLINDPIXEL mask is set completely to false because all pixels are read. This could change in the future.

7 Level 0 to Level 0.5

We assume that the reader is starting the data reduction from Level 0 Products. Tasks and procedures related to creation of pools from telemetry files or from database need a deeper knowledge of the system and are included at the end of this chapter. The PACS Photometer pipeline is composed of tasks written in java and jython. In this section we explain the individual steps of the pipeline up to Level 0.5. Up to this product level the data reduction is AOT independent. The only AOT dependent task executed in this part of the data reduction is the CleanPlateau task, which is executed only for chopped observations

(Point-Source, Small source and Chopped Raster AOT).

7.1 Getting started: how to retrieve data in the Observation Context

We assume that the reader got a tar file containing all the chosen observations and associated data from the HSA. Unpacking this tar file should automatically create a 'so called' local store with one or more pools. Any pool contains a number of directories containing data products of different level (Level 0,1,2) and calibration files for each observation. A special pool contains the Auxiliary products with, in particular, the Pointing and the SIAM products which are needed for the astrometric calibration. The Pointing Product contains all the info about the satellite pointing and the SIAM Product provides info abot the orientation of the PACS detectors with respect to the satellite. We list here few commands that the user needs to execute to retrieve a given observation from a pool and start the data reduction:

```
lstore = LocalStoreFactory.getStore("test_pool")
store  = ProductStorage()
store.register(lstore)
result = browseProduct(store)
#in alternative:
query=MetaQuery(ObservationContext,'h','h.meta["obsid"].value == 32212260061')
result=store.select(query)
```

The first three commands listed in the window above access and register a test pool ("test_pool"). The fourth command calls the Product Browser to inspect the content of the Observation Context and choose a given observation. We refer the reader to Chapter 12.1.10 for a detailed description of the Product Browser and its use. The observation chosen in the Product Browser is, then, stored in the variable "result". In alternative, if the content of the pool is already known, we can query a particular observation on the basis of its OBSID, which is a unique identification number. The result of the query is, then, stored in the result variable as done in the case of the Product Browser.

```
obs=result[0].product
frames=obs.level0.refs["HPPAVGB"].product.refs[0].product
hkdata = obs.level0.refs["HPPHK"].product.refs[0].product["HPPHKS"]
pp=obs.auxiliary.pointing
calTree = getCalTree("FM", "BASE")
```

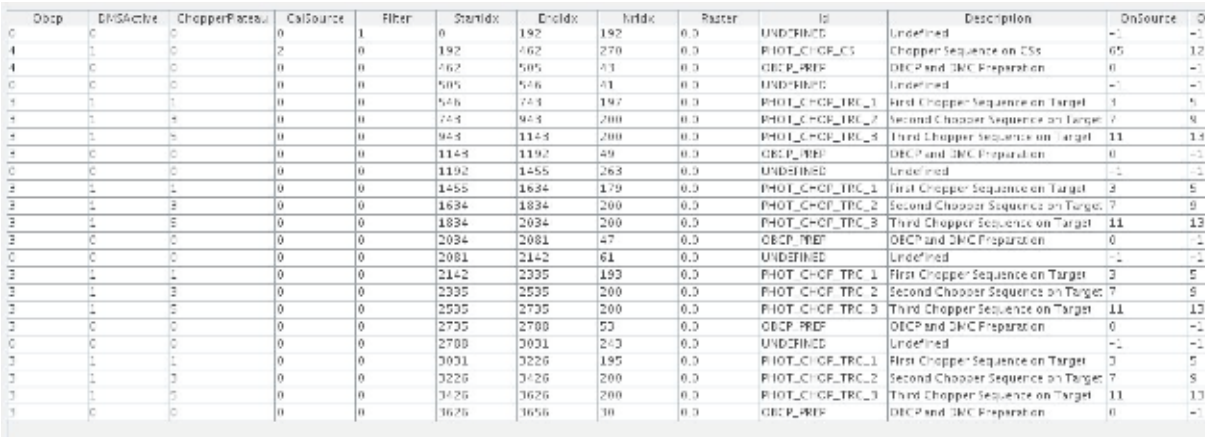
After selecting our favorite observation, we can store a given product (Level 0, 1 or 2 if they exist in the considered pool) in the "obs" variable, as shown in the first command in the window above. In our example we select all the information relative to the Level 0 product to start with the first step of the data processing. The second command selects the frames of the Blue bolometer and store them in the variable "frames", a frames class. The string "HPPAVGB" needs to be changed to "HPPAVGR" to select products of the Red bolometer. The frames class is composed of a data cube containing all the readouts of our observation, the so-called Status table with several entries for each readout, the BLIND-PIXEL mask and several metadata. With a similar syntax we store in the "hk" variable several House Keeping values of our observation. This variable will be used directly in the next step of the data reduction. This info is needed for the further data calibration. The last two commands store the Pointing Product in the variable "pp" and the Calibration Tree in the calTree variable. The Calibration Tree is a way to organize all the calibration files attached to our observation. It allows us to specify only the version of the calibration file to use, without specifying for any task the file name.

7.2 The second step, understanding what there is in the observation: findBlocks (jython prototype available)

```
>> outFrames = findBlocks(inFrames)

outFrames : Frames : Frames out
inFrames  : Frames : Frames in
```

This task is not essential for the data reduction. We can reduce the data even without executing the task findblock. However, we suggest to execute this task just after getting the data to understand what there is

in the dataset. The result of this task is a table, called BlockTable, containing info about the structure of the observation. Any observation is a composition of observing block (e.g. calibration block and science data block). The BlockTable contains info, for instance, about how many calibration blocks were executed during the observation, how many chop-nod cycles or scan legs are contained in the data, etc. Basically, the BlockTable summarizes, per observation block, several info already contained, per readout, in the Status Table. This info can help us in checking if the data contains the observation as we have designed it, in selecting just part of the observation for a preliminary data reduction, in slicing the data as we desire, etc.



Obcp	DMSActive	ChopperPlateau	CalSource	Filter	StartIdx	EndIdx	NrIdx	Raster	Id	Description	OnSource	O
0	0	0	0	1	0	192	192	0.0	UNDEFINED	Undefined	-1	-1
4	1	0	2	0	192	462	270	0.0	PHOT_CHOP_CS	Chopper Sequence on CSs	65	12
4	0	0	0	0	462	505	43	0.0	OBCP_PREP	OBCP and DMC Preparation	0	-1
0	0	0	0	0	505	546	41	0.0	UNDEFINED	Undefined	-1	-1
1	1	1	0	0	546	744	197	0.0	PHOT_CHOP_TRG_1	First Chopper Sequence on Target	3	5
1	1	1	0	0	744	944	200	0.0	PHOT_CHOP_TRG_2	Second Chopper Sequence on Target	7	9
1	1	1	0	0	944	1144	200	0.0	PHOT_CHOP_TRG_3	Third Chopper Sequence on Target	11	13
1	0	0	0	0	1144	1192	49	0.0	OBCP_PREP	OBCP and DMC Preparation	0	-1
0	0	0	0	0	1192	1455	263	0.0	UNDEFINED	Undefined	-1	-1
3	1	1	0	0	1455	1634	179	0.0	PHOT_CHOP_TRG_1	First Chopper Sequence on Target	3	5
3	1	1	0	0	1634	1834	200	0.0	PHOT_CHOP_TRG_2	Second Chopper Sequence on Target	7	9
3	1	1	0	0	1834	2034	200	0.0	PHOT_CHOP_TRG_3	Third Chopper Sequence on Target	11	13
3	0	0	0	0	2034	2081	47	0.0	OBCP_PREP	OBCP and DMC Preparation	0	-1
0	0	0	0	0	2081	2142	61	0.0	UNDEFINED	Undefined	-1	-1
3	1	1	0	0	2142	2335	193	0.0	PHOT_CHOP_TRG_1	First Chopper Sequence on Target	3	5
3	1	1	0	0	2335	2535	200	0.0	PHOT_CHOP_TRG_2	Second Chopper Sequence on Target	7	9
3	1	1	0	0	2535	2735	200	0.0	PHOT_CHOP_TRG_3	Third Chopper Sequence on Target	11	13
3	0	0	0	0	2735	2788	53	0.0	OBCP_PREP	OBCP and DMC Preparation	0	-1
0	0	0	0	0	2788	3031	243	0.0	UNDEFINED	Undefined	-1	-1
3	1	1	0	0	3031	3226	195	0.0	PHOT_CHOP_TRG_1	First Chopper Sequence on Target	3	5
3	1	1	0	0	3226	3426	200	0.0	PHOT_CHOP_TRG_2	Second Chopper Sequence on Target	7	9
3	1	1	0	0	3426	3626	200	0.0	PHOT_CHOP_TRG_3	Third Chopper Sequence on Target	11	13
3	0	0	0	0	3626	3656	30	0.0	OBCP_PREP	OBCP and DMC Preparation	0	-1

The figure above shows a BlockTable example for a chopped observation, in particular a point-source AOT in the jide DataSetInspector (a similar layout can be obtained in the hipe Editor). The main ingredient of the BlockTable is the OBCP/DMC number (first column in the example above). This number identifies what PACS photometer is doing for a given time, that is between the time indexes StartIdx and EndIdx of our observation. A verbal translation of the OBCP/DMC number is given in the Id and Description columns. For instance, we can easily see in which part the observation the calibration block was executed (id="PHOT_CHOP_CS"), or when the PACS photometer is preparing itself fo the next command (Id="OBCP and DMC preparation" or Id="Undefined") and when the real scientific data are taken and the first chopper sequence is executed (Id="PHOT_CHOP_TRG_1"). Satellite pointing information/mode (staring, nod-position A or B, raster point M-N, scan leg number, tracking, hold position, etc.) can also be included in the BlockTable if findBlock is executed after the execution of the AddInstantPointing task which adds the pointing information to the frames class. Typical AOT observations might contain several OBCPs and some of the OBCPs might be executed many times within one AOT observation (as in the example above). A block contained in a given OBCP/DMC sequence might correspond to different labels. In some cases a change in label does not necessarily mean a change in chopper position, e.g. see below, block 2 in OBCP 04 "chopper photometry sequence on target" can have labels 1,3, and 5. Here, labels 3 and 5 correspond very likely to the same chopper position. We list here the OBCP/DMC sequences and the blocks associated to them (taken from PACS-ME-LI-005, Issue 1.1, 08-Mar-2005) :

- OBCP 01: Bolometer transition to IDLE state ---> no DMC sequence, no blocks
- OBCP 02: Bolometer operation for unregulated state ---> no DMC sequence, no blocks
- OBCP 03: Fixed-Fixed Chopped Photometry ---> DMC sequence 14 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - first chopper sequence on target (labels 3,5)
  - second chopper sequence on target (labels 7,9)
  - third chopper sequence on target (labels 11,13)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 04: Chopped Photometry ---> DMC sequence 1 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - chopper sequence on target (labels 1,3,5)
  - chopper sequence on CSs (labels 65,129)
  - DMC and OBCP Reset (label 0, undefined)

- OBCP 05: Chopped Photometry with Dither ---> DMC sequence 2 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - chopper sequence on target (labels 1,3,5)
  - chopper sequence on CSs (labels 65,129)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 06: Freeze Frame Chopping Photometry ---> DMC sequence 4 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - freeze frame sequence on target (label 63)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 07: Staring Photometry for Line Scans ---> DMC sequence 3 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - staring sequence on target (label 1)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 08: Grating Spectral Line Scan Chopped ---> DMC sequence 8 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - chopped up-scan sequence on target (labels 3,5,7)
  - chopped up-scan sequence on CSs (labels 65,129)
  - chopped down-scan sequence on target (labels 19,21,23)
  - chopped down-scan sequence on CSs (labels 81,145)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 09: Grating Spectral Line Scan Chopped with Dither ---> DMC sequence 9 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - chopped up-scan sequence on target (labels 3,5,7)
  - chopped up-scan sequence on CSs (labels 65,129)
  - chopped down-scan sequence on target (labels 19,21,23)
  - chopped down-scan sequence on CSs (labels 81,145)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 10: Photometry Calibration I ---> DMC sequence 5 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - variable-variable chopped sequence on CSs (labels 65, 129)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 11: Photometry Calibration II ---> DMC sequence 6 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - fixed-variable chopped sequence on CSs (labels 65, 129)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 12: Photometry Calibration III ---> DMC sequence 7 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - fixed-fixed chopped sequence on CSs (labels 65, 129)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 13: Internal Calibration Spectroscopy ---> DMC sequence 11 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - chopped up-scan sequence on CSs (labels 65,129)
  - chopped down-scan sequence on CSs (labels 81,145)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 14: Acquire Non-Sequencer Science Data ---> no DMC sequence, no blocks
- OBCP 15: DMC Test Mode ---> no DMC sequence, no blocks
- OBCP 16: Switch Spectroscopy to Photometry ---> no DMC sequence, no blocks
- OBCP 17: Switch Photometry to Spectroscopy ---> no DMC sequence, no blocks
- OBCP 18: Prepare for Switch-off ---> no DMC sequence, no blocks
- OBCP 19: Start 1355 link ---> no DMC sequence, no blocks
- OBCP 20: Write in EEPROM ---> no DMC sequence, no blocks

- OBCP 21: Start HLSW ---> no DMC sequence, no blocks
- OBCP 22: Wavelength Switch Grating ---> DMC sequence 10 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - first grating switch sequence (label 33)
  - second grating switch sequence (label 97)
  - third grating switch sequence (label 161)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 23: Ge:Ga Set-up ---> no DMC sequence, no blocks
- OBCP 24: Switch to SAFE ---> no DMC sequence, no blocks
- OBCP 25: Time Synchronisation Test 1 ---> no DMC sequence, no blocks
- OBCP 26: Time Synchronisation Test 2 ---> no DMC sequence, no blocks
- OBCP 27: Grating Line Scan Chopped 2 ---> DMC sequence 12 blocks:
  - OBCP and DMC preparation (label undefined, 0)
  - chopped up-scan sequence on target (labels 3,5)
  - chopped up-scan sequence on CSs (labels 65,129)
  - chopped down-scan sequence on target (labels 19,21)
  - chopped down-scan sequence on CSs (labels 81,145)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 28: Grating Line Scan Without Chopping ---> DMC sequence 13 blocks
  - OBCP and DMC preparation (label undefined, 0)
  - up-scan sequence on target + CSs (labels 3,65,129)
  - down-scan sequence on target + CSs (labels 19,81,145)
  - DMC and OBCP Reset (label 0, undefined)
- OBCP 29: Generate dummy science packets ---> no DMC sequence, no blocks
- OBCP 30: SPU test mode SPEC ---> no DMC sequence, no blocks
- OBCP 31: SPU test mode PHOT ---> no DMC sequence, no blocks
- OBCP 32: BION ---> no DMC sequence, no blocks
- OBCP 33: OBMO ---> no DMC sequence, no blocks
- OBCP 34: ACWE ---> no DMC sequence, no blocks

7.3 Pre-processing of the calibration blocks

In the current observation design strategy a calibration block is executed at the beginning of any observation. It is possible that in the future the current design will be changed to include more than one calibration block to be executed at different times during the observation. In order to take into account this possible change, the pipeline includes as a very first step a pre-processing of the calibration block(s) that is planned to work under any possible calibration block(configuration. The calibration block pre-processing is done in three steps: a) the calibration block(s) is identified and extracted from the frames class, b) it is reduced by using appropriate and pre-existing pipeline steps, c) the result of the cal block data reduction is attached to the frames class to be used in the further steps of the data reduction.

7.3.1 photCSExtraction (prototype to be tested)

This is the first step of the calibration block pre-processing. This task identifies the calibration block(s) of the given observation and it stores it in an additional frames class (csFrames in the example below). This is still to be clarified. At the moment the cal block is stored in a context, but this should not be necessary. The input frames remain unchanged.

```
>> MapContext csFrames = photCSExtraction(Frames inFrames)
csFrames : MapContext : list of frames containing calibration blocks.
                        One block per frame.
inFrames : Frames     : Frames in ↵
```

7.3.2 photCSProcessing (prototype to be tested)

Once the calibration block is identified and stored in a proper frames class the calibration data can be

reduced. The input of this task are the frames containing the calibration block (csFrames of previous task), the House Keeping (hkdata as in section 7.1), and, as additional input the packet sequence "seq". However, the packet sequence will not be available to the end-user. So this additional entry is already obsolete and it will be soon removed at the end of the testing phase. The output of the task is the differential image of the two calibration sources plus several House Keeping values extracted from the hkdata variable. Those info are necessary to correct any drift in the flat-field and flux calibration (see sections related to this tasks for more details).

```
>> MapContext csFrames = photCSProcessing(Frames inFrames, ArrayDataset hkdata,
                 PacketSequence seq)
csFrames : MapContext     : list of processed frames containign calibration blocks.
inFrames : MapContext     : list of raw calibration blocks - the result of the module
                            photCSExtraction
hkdata   : TableDataset   : housekeeping information extracted from the observation
                            context
seq      : packetSequence : packet sequence containing housekeeping data - to use if
                            hkdata is missing (valid only for the testing phase)
```

Since the calibration block is nothing else than a chopped observation, the calibration data are reduced in analogy to the Point-source data. Thus, this module call all the remaining tasks described in the current section up to level 0.5 and few specific tasks of the Point-source pipeline between level 0.5 and 1. We list below the tasks called in the execution of this module: - photFlagBadPixels adds badpixel mask to the frames - photFlagSaturation adds a mask containing pixels saturated according to the bolometer settings. - photConvDigit2Volts converts the calibration block signals into Volt - photCorrectCrossTalk corrects the crosstalk of each pixel - photDeglitching flags/removes glitches found in the signal - photCleanPlateau identifies chopper plateau at the calibration source - photAvgPlateau calculates the average of each plateau - photDiffCal calculates the differential image of the two calibration sources CS1-CS2, in addition it collects the housekeeping data relative to the calibration, such as gain and bias settings of the observation.

Output : after the execution of this task each frames stored in csFrames contains three more pieces of information :

```
-DCs differential image of the calibration sources (CS1-CS2)
-DCsNoise noise image of the CS1-CS2 subtraction
-HkCalBlkTable a TableDataset containing housekeeping data relative to
the calibration blocks
```

We point out that also this task and the previous one is still in a testing phase. DCs, DCsNoise, HkCalBlkTable are used by photDriftCorrection module.

7.3.2.1 photDiffCal

Among the tasks used for the reduction of the calibration block and listed above, only photDiffCal is specific to this case. All the other tasks will be described in the point-source and small-source pipelines. Here we describe what this task is doing in detail. photDiffCal adds a TableDataset named HkCalBlkTable to the frame. For each calibration block encountered new rows is added to the table. Each column contains one type of information such as the CPR position, start time and end time of the calibration block and so on). This table is reused later on by photDriftCorrection and photDiffCStoringTask. Here is briefly the name of the columns:

- Index : calibration block index for this obsid
- Channel : channel currently processed by the pipeline
- StartTime : start time of the calibration block (finetime)
- EndTime : end time of the calibration block (finetime)
- MeanTime : mean time of the calibration block (finetime)
- CS1-Temp : average of CS1 temperature found inside the calibration block
- CS2-Temp : average of CS2 temperature found inside the calibration block

- CS1-Sigma : gives the standard deviation of the temperature of CS1 for the currently calibration block processed
- CS2-Sigma : gives the standard deviation of the temperature of CS2 for the currently calibration block processed
- CS1-CPR : the average of the chopper position when PACS looks at its first calibration source
- CS2-CPR : the average of the chopper position when PACS looks at its second calibration source
- Mode : median of the readout mode (Direct or DDCS)
- Bias : average of <VH-VL> for all BU
- Gain : low or high (1 or 0)
- Frequency : velocity of the chopping between CS1 and CS2 in Hz
- Filename : possible reusable file name "PTrendPhotometer_diffCS_'Date=YYMMDD_hhmmss'_FM_"

PhotDiffCal adds as well two additional computations to the frame : the difference of the calibration source (indexed with 'DCs' keyword) and the noise computed for each calibration block (indexed with 'DCsNoise' keyword). Here are the formulas used to compute the noise "DCsNoise" :

$$DCsNoise=SQRT(<dnoise^2>)/SQRT(n)$$

where dnoise is the noise of the individual CS1-CS2, cal sources differential images, involved in the average. The individual dnoise are obtained as the sum in quadratur of the CS1 and CS2 noises as computed by the the previous photAvgPlateau task. n is the number of samples of cs1-cs2 measurements averaged on the calibration block interval.

7.3.3 photCSClean (prototype to be tested)

We pointed out in the description of the PhotCSExtraction task, that the input frames remain unchanged. That frames is the class where we stored all the Level 0 products, including the calibration block. Since we already reduced the cal block in the previous task and we extracted the useful info from it, we now want to remove it from the original frames in order to keep only the scientific data and to store only the essential information about the calibration sources. Thus, this task simply removes the cal block from the original frames and replaces it with the output of the previous task. The output frames will contain the scientific data plus the DCs and DCsNoise images and HkCalBlkTable (see previous task).

```
>> outframes = photCSClean(Frames inFrames)

outFrames : Frames     : Frames out
inFrames  : Frames     : Frames in
csFrames  : MapContext : calibration block encapsulated into frame -
            result of photCSProcessing module
```

7.4 photFlagBadPixels

The purpose of this task is to flag the damaged pixels in the BADPIXEL mask. The task should do a twofold job: a) reading the existing bad pixel mask provided by a calibration file ("PCalPhotometer_BadPixelMask_FM_v1.fits" in the current release), b) identifying additional bad pixels during the observation. In the current version of the pipeline only the first functionality is activated. The algorithm for the identification of additional bad pixels is not in place. So the task is just reading the bad pixel calibration file and transforming the 2D mask contained in it in the 3D BADPIXEL mask. The task is doing the same for the BLINDPIXEL mask. This is an uplink mask, which currently is completely set to false. The purpose is to use it to indicate the pixels which should not be read at all and for which data should not be downloaded.

```
>> outFrames = photFlagBadPixels(inFrames [,calTree=calTree] [,copy=copy] )


outFrames  : Frames : Frames out
inFrames   : Frames : Frames in
calTree    : PacsCalibrationTree : calibration tree containing all calibration
```

```
              products used by the pipeline
copy        : int    :
                      0 - return reference
                      1 - return copy
```

7.5 photFlagSaturation

This tasks identifies the saturated pixels on the basis of saturation limits contained in a calibration file. Before doing that, the task identifies the reading mode led by the warm electronic BOLC (Direct or DDCS mode) and the gain (low or high) used during the observation. These information are provided for each sample of the science frames by the BOLST entry in the status table. The task compares the pixel signal at any time index to the dynamic range corresponding to the identified combination of reading mode and gain. Readout values above the saturation limit are flagged in the 3D SATURATION mask.

```
>> outFrames = photFlagSaturation(inFrames [,calTree=calTree] [,satLimits=satLimits]
                       [,copy=copy])

outFrames   : Frames : Frames out
inFrames    : Frames : Frames in
calTree     : PacsCalibrationTree : calibration tree containing all calibration products
              used by the pipeline
satLimits   : Double1D, in case the user passes the satLimits explicitly
copy        : int    :
                      0 - return reference
                      1 - return copy
```

7.6 photConvDigit2Volts

The task converts the digital readouts to Volts. As in the previous task, as a first step the task identifies the reading mode and the gain on the basis of the the BOLST entry in the status table for each sample of the frame. This is redundant and this step will be skipped when mode and gain will be stored in the metadata of the Level 0 Product. The task extracts, then, the appropriate value of the gain (high or low) and the corresponding offset (positive for the direct mode and negative for the DDCS mode) from the calibration file (PCalPhotometer_Gain_FM_v1.fits in the current release). These values are used in the following formula to convert the signal from digital units to volts: signal(volts) = (signal(ADU) - offset) * gain

```
>> outFrames = photConvDigit2Volts(inFrames [,calTree=calTree] [,photGain=photgain]
                       [,copy=copy])

outFrames   : Frames : Frames out
inFrames    : Frames : Frames in
calTree     : PacsCalibrationTree : calibration tree containing all calibration products
              used by the pipeline
photGain    : gain   : nominal gain (1, 100 uV/step), low gain (5, 20 uV/step)
              or high gain(20, 5uV/step)
copy        : int    :
                      0 - return reference
                      1 - return copy
```

Reference : BOLC TO DMC ELECTRICAL INTERFACE CONTROL DOCUMENT (SAp-PACS-cca-0046-01)

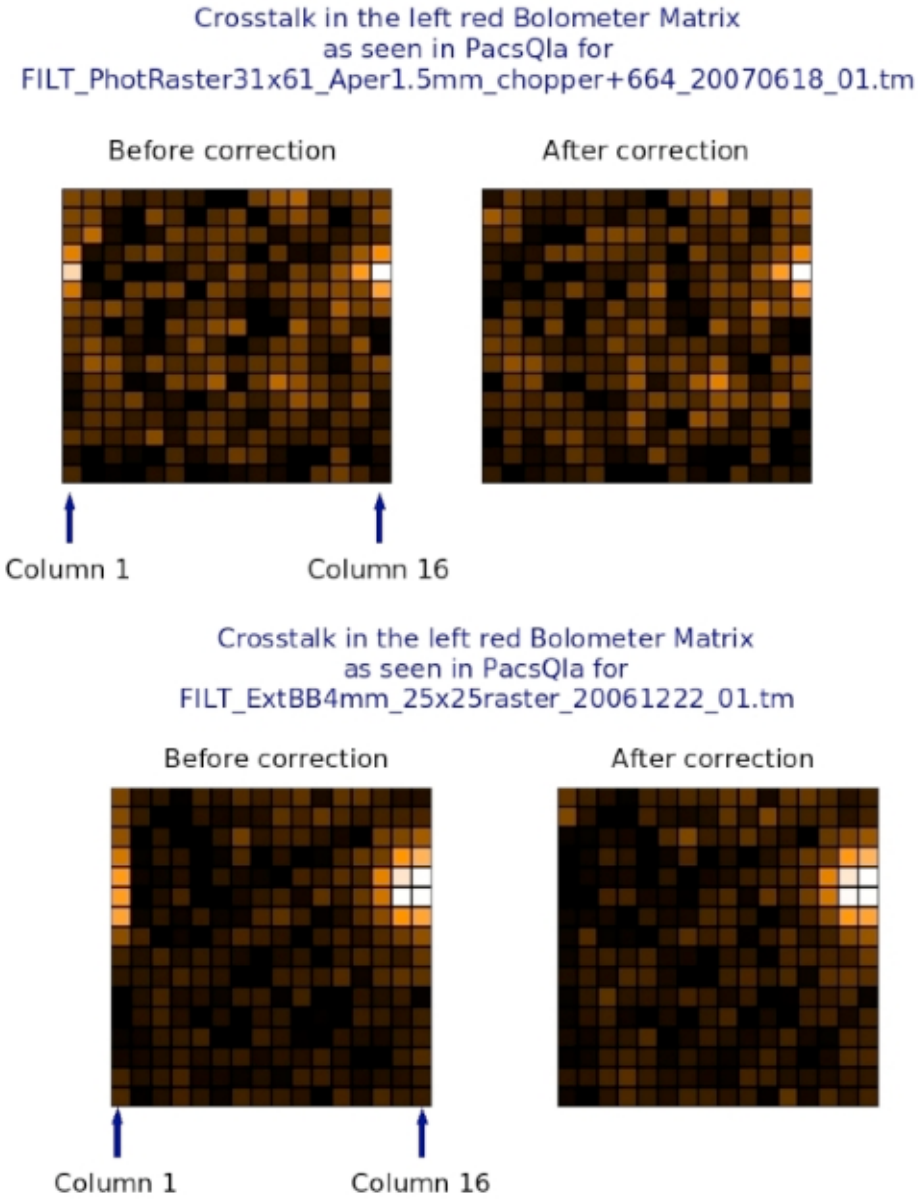
7.7 photCorrectCrosstalk

The phenomenon of electronic crosstalk was identified, in particular in the red bolometer, during the testing phase. The working hypothesis of this task is that the amount of signal in the crosstalking pixel is a fixed percentage of the signal of the correlated pixel. A calibration file (PCal_PhotometerCrosstalkMatrix_FM_v2.fits in the current release) reports a table containing the coordinates of crosstalking and correlated pixels and the percentage of signal to be removed, for the red and the blue bolometer, respectively, . The task reads the calibration file and use the info stored in the appropriate table to apply the following formula:

Signal_correct(crosstalking pixel)) = Signal(crosstalking pixel) - a*Signal(correlated pixel)

where 'a' is the percentage of signal of the correlated pixel to be removed from the signal of the crosstalking pixel.The task is still under investigation, in the sense that invariability of 'a' is still an assumption to be tested in further tests.

Crosstalk before and after the crosstalk correction task has been applied



In the above two images we show two examples of electronic crosstalk in the red bolometer for different source fluxes. The left side shows the situation before the correction. The right side shows the result after the correction. The task removes succefully the fraction of the signal in column 1 due to the correlated column 16. However, it is worth to notice that in the second case, the crosstalk is somehow over-corrected. This would imply that 'a' could depend also on the signal of the correlated pixel. Moreover it is known that the amount of crosstalk can be influenced also by the photometer bias voltage settings. Future tests are planned to explore all these possibilities, in particular, a) finding a bias settings able to minimize/avoid the crosstalk or, in alternative, b) studying the dependence of 'a' on the bias settings and providing a new calibration file which takes into account this dependence.

```
>> outFrames = photCorrectCrosstalk(inFrames [,copy=copy])
```

```
outFrames   : Frames : Frames out
inFrames    : Frames : Frames in
copy        : int    :
                      0 - (False) return reference
                      1 - (True) return copy
```

Reference: D. Lutz, P. Popesso. Bolometer Spatial Calibration, PACS Test Analysis Report FM-ILT Version 0.0 from October 25/2007

7.8 photMMTDeglitching and photWTMMLDeglitching

These tasks detect, mask and remove the effects of cosmic rays on the bolometer. Two different tasks are implemented for the same purpose: photMMTDeglitching is based on the multiresolution median transforms (MMT) proposed by Starck et al (1996), WTMMLDeglitching is based on the Wavelet Transform Modulus Maxima Lines Analysis (WTMML). The former task is in the testing phase. The tests aim at identifying suitable ranges of parameters for different scientific cases. The latter task is still under investigation and debugging phase. At this stage of the data reduction the astrometric calibration has still to be performed. Thus, the two tasks can not be based on redundancy. Both tasks have to overcome the following problems:

- signal fluctuation of each pixel,
- the movement of the telescope,
- the hits received by one pixel due to several cosmic rays having different signatures and arrival time,
- the non-linear nature of each glitch.

7.8.1 Deglitching using the Multiresolution Median Transform (photMMTDeglitching)

This task is based on the method developed by Starck et al. (1998) for the detection of faint sources in ISOCAM data. The method relies on the fact that the signal due to a real source and to a glich, respectively, when measured by a pixel, shows different signatures in its temporal evolution and can be identified using a multiscale transform which separates the various frequencies in the signal. Once the "bad" components due to the glitches are identified, they can be corrected in the temporal signal. Basically, the method is based on the multiresolution support. We say that a multiresolution support (Starck et al. 1995) of an image describes in a logical or boolean way if an image f contains information at a given scale j and at a given position (x,y). If the multiresolution support of f is M(j,x,y)=1 (or true), then f contains information at scale j and position (x,y). The way to create a multiresolution support is trough the wavelet transform. The wavelet transform is obtained by using the multiresolution median transform. The median transform is nonlinear and offers advantages for robust smoothing. Define the median transform of an image f, whit the square kernel of dimension n x n, as med(f,n). Let n=2s+1; initially s=1. The iteration counter will be denoted by j, and S is the user-specified number of resolution scales. The multiresolution median transform is obtained in the following way:

1. Let $c_j = f$ with $j = 1$.
2. Determine $c_{j+1} = \mathrm{med}(f, 2s+1)$.
3. The multiresolution coefficients w_{j+1} are defined as: $w_{j+1} = c_j - c_{j+1}$.
4. Let $j \longleftarrow j+1$; $s \longleftarrow 2s$. Return to step 2 if $j < S$.

A straightforward expansion formula for the original image (per pixel) is given by:

$$c_0(x,y) = c_p(x,y) + \sum_{j=1}^{p} w_j(x,y).$$

where, cp is the residual image. The multiresolution support is obtained by detecting at each scale j the significant coefficient wj. The multiresolution support is defined by:

$$M(j,x,y) = \begin{cases} 1, & \text{if } w_j(x,y) \text{ is significant;} \\ 0, & \text{if } w_j(x,y) \text{ is not significant.} \end{cases}$$

Given stationary Gaussian noise, the significance of the w_j coefficients is set by the following conditions:

$$\text{if } |w_j| \geq k\sigma_j, \text{ then } w_j \text{ is significant;}$$

$$\text{if } |w_j| < k\sigma_j, \text{ then } w_j \text{ is not significant.}$$

where sigma_j is the standard deviation at the scale j and k is a factor, often chosen as 3. The appropriate value of sigma_j in the succession of the wavelet planes is assessed from the standard deviation of the noise, sigma_f, in the original f image. The study of the properties of the wavelet transform in case of Gaussian noise, reveals that sigma_j=sigma_f*sigma_jG, where sigma_jG is the standard deviation at each scale of the wavelet transform of an image containing only Gaussian noise. The standard deviation of the noise at scale j of the image is equal to the standard deviation of the noise of the image multiplied by the standard deviation of the noise of the scale j of the wavelet transform. In order to properly calculate the standard deviation of the noise and, thus, the significant wj coefficients, the tasks applies an iterative method, as done in starck et al. 1998:

- calculate the Multiresolution Median Transform of the signal for every pixel
- calculate a first guess of the image noise. The noise is estimated using a MeanFilter with boxsize 3 (Olsens et al. 1993)
- calculate the standard deviation of the noise estimate
- calculate a first estimate of the noise in the wavelet space
- the standard deviation of the noise in the wavelet space of the image is then sigma(j) = sigma(f)*sigma(jG) (Starck 1998).
- the multiresolution support is calculated
- the image noise is recalculated over the pixels with M(j,x,y)=0 (containing only noise)
- the standard deviation of the noise in the wavelet space, the multiresolution support and the image noise are recalculated iteratively till (noise(n) - noise(n-1))/noise(n) <**noiseDetectionLimit**, where noiseDetectionLimit is a user specified parameter (Note: if your image does not contain pixels with only noise, this algorithm may not converge. The same is true, if the value **noiseDetectionLimit** is not well chosen. In this case the pixel with the smallest signal is taken and treated as if it were noise)

At the end of the iteration, the final multiresolution support is obtained. This is used to identify the significant coefficients and , thus, the pixels and scales of the significant signal. Of course, this identifies both glitches and real sources. According to Starck et al. (1998), at this stage a pattern recognition should be applied in order to separate the glitch from the real source components. This is done on the basis of the knowledge of the detector behavior when hit by a glitch and of the different effects caused in the wavelet space by the different glitches (short features, faders and dippers, see Starck at al. 1998 for more details). This knowledge is still not available for the PACS detectors. At the moment, a real pattern recognition is not applied and the only way to isolate glitches from real sources is to properly set the user-defined parameter **scales** (S in the description of the multiresolution median transform above). The method works reasonably well till the maximum number of readouts of a glitch is much smaller than the one of a real source (**scales** < 5, see the alpha irradiation example below). For higher value of **scales** (< 5), also part of the signal of a real bright source can be identified as a glitch (see proton irradiation example below).

When the glitches are identified the signal of the pixel affected is corrected by interpolating the signal before and after the glitch event. In addition, the task also produces the 3D MMTGLITCH mask which flags the deglitched pixels at any time (due to a bug of the task the MMTGLITCH mask is not always produced, this is under investigation). However, it is required that the task does not correct by default the signal of the pixels affected by glitches. It is foreseen that the task will provide the user the possibility to choose

whether to correct or not the signal or to have only the MMTGLITCH mask as a result.

Literature reference for this algorithm: ISOCAM Data Processing, Stark, Abergel, Aussel, Sauvage, Gastaud et. al., Astron. Astrophys. Suppl. Ser. **134**, 135-148 (1999) Automatic Noise Estimation from the Multiresolution Support, Starck, Murtagh, PASP, **110**, 193-199 (1998) Estimation of Noise in Images: An Evaluation, Olsen, Graphical Models and Image Processing, **55**, 319-323 (1993)

7.8.1.1 Details and Results of the implementation

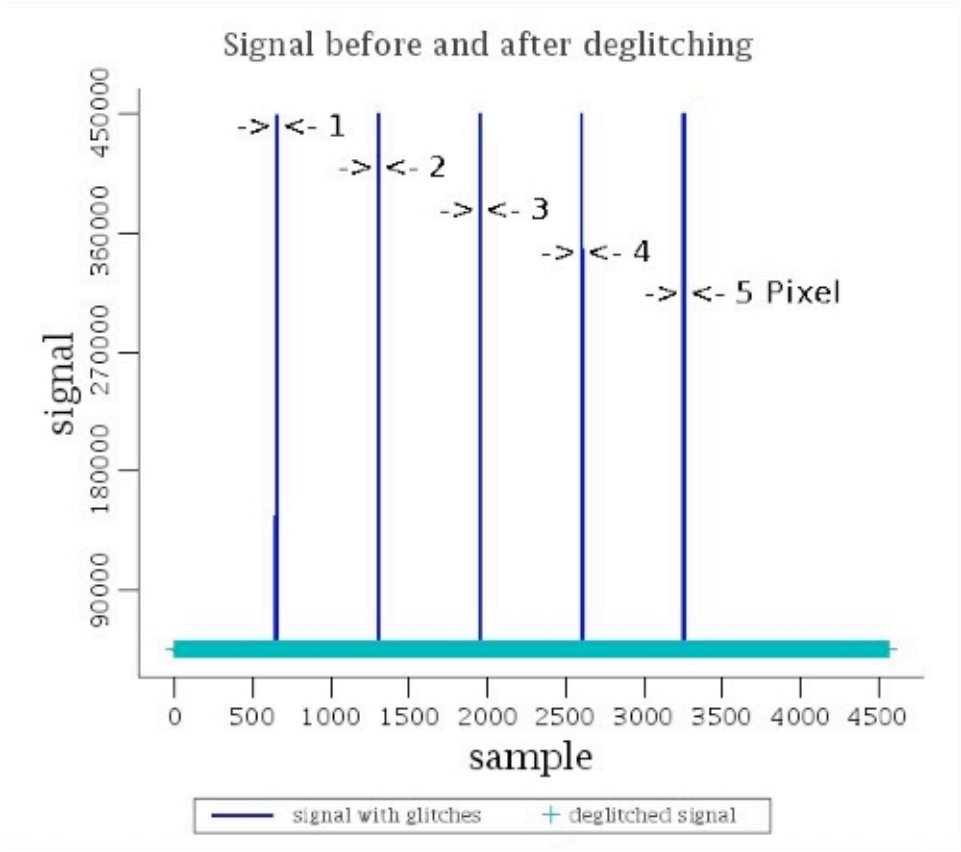
This is the signature of the task:

```
>> outframes = photMMTDeglitching(inFrames [,copy=copy] [,scales=scales]
                          [,mmt_startenv=mmt_startenv] [,incr/fact=incr/fact]
                          [,mmt_mode=mmt_mode] [,mmt_scales=mmt_scales]
                          [,nsigma=nsigma])

outFrames  : the returned Frames object
inFrames   : the Frames object with the data that should be deglitched
copy       : boolean. Possible values: false (jython: 0) - inFrames will be modified
             and returned true (jython: 1) - a copy of inFrames will be returned

scales     : int. Number of wavelet scales. This should reflect the maximum expected
             readout number of the glitches. Default is 5 readouts.
mmt_startenv : int. The startsize of the environment box for the median transform.
               Default is 1 readout (plus/minus).
incr/fact  : float. Increment resp. factor to enhance the mmt_startenv. Default is 1.
mmt_mode   : int. Defines how the environment should be modified between the scales.
             Possible values: 1 (ADD) or 0 (Multiply). Default is 1.
             example: the environment-size for the subsequent median transform
             environment boxes will be  env(0) = mmt_startenv, env(n) = env(n-1)
             mmt_mode incr/fact default means then: env(0) = 1, env(1) = 1 + 1,
             env(2) = 3 etc.
noiseDetectionLimit : double. Threshold for determining the image noise. values between
                      0.0 and 1.0. Default is 0.3.
nsigma     : int. Limit that defines the glitches on the wavelet level. Every value
             larger than nsigma*sigma will be treated as glitch. Default is 5.
```

The default values of the parameters specified above do not provide a good default for the different scientific cases. After several tests on different AOT test-cases, the 'scales' parameter tuns out to be the most relevant for a good performace of the method. For scan map observation the 'scales' parameter can be rather safely set to 2-3. In the case of chopped observations a good value is scales=1. We point out that photMMTDeglitching is not a good choice for masking out glitched readouts in chopped observation. Indeed, the relatively short chopper plateaus might be wrongly identified by this method as glitches. An alternative method, based on a simple sigma clipping in one of the chopped data pipeline tasks can probably deal more efficiently with glitches than the method described here. However, we point out that only in orbit tests can provide reliable suggestions for the different scientific cases.

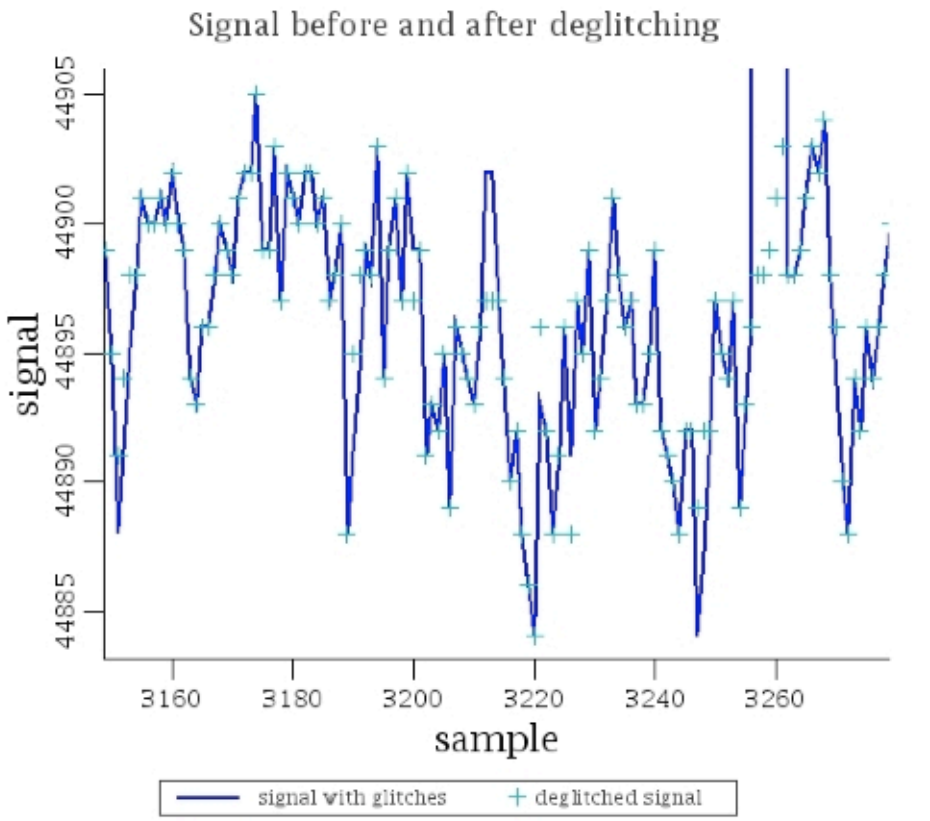
7.8.1.2 Results of example data

To examine the result of this algorithm, a real Bolometer signal has been taken and artificial glitches with a width of 1, 2, 3, 4 and 5 pixels have been added to the signal of one of the pixels. The analysis has then been done with 6 wavelet scales.

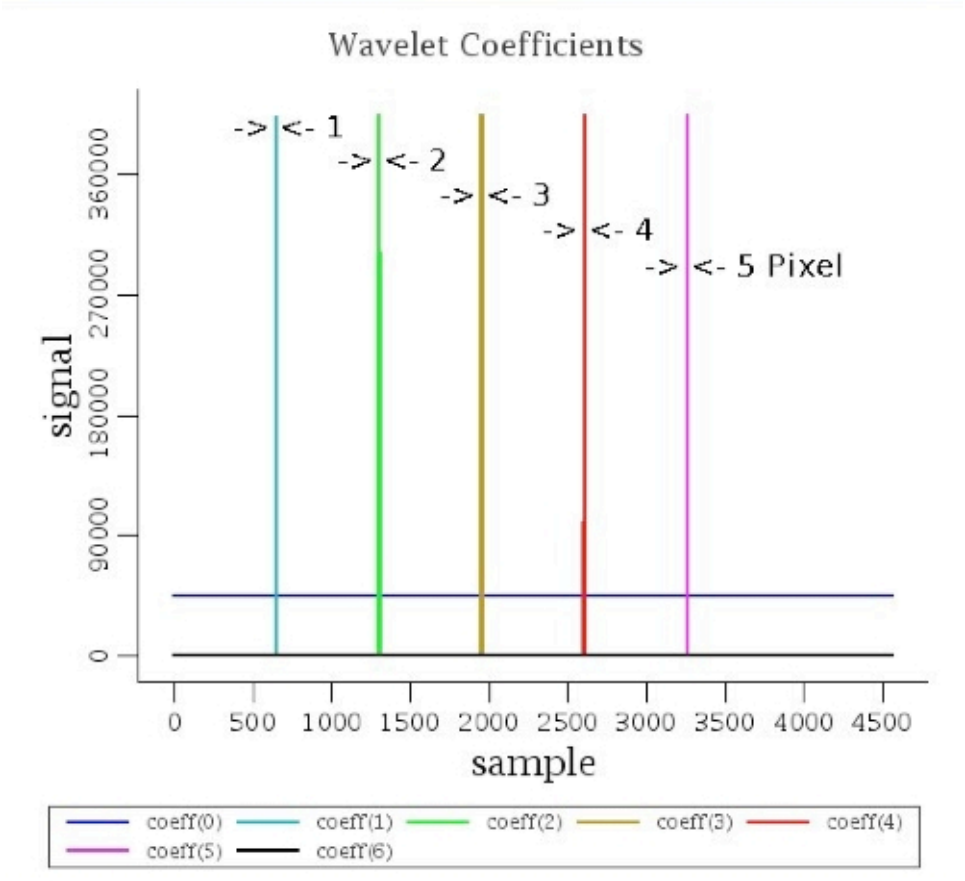


Signal with glitches and deglitched signal

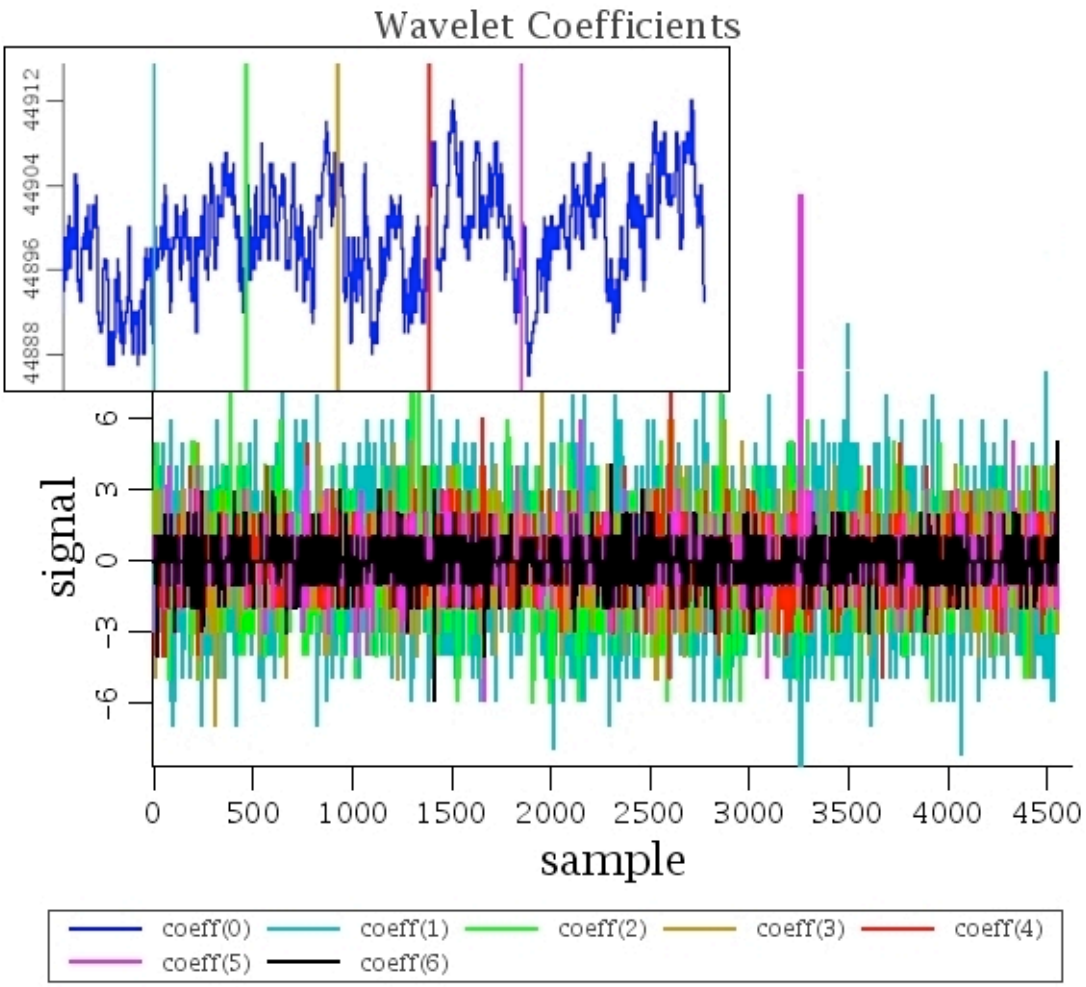
A closer look at the signal and the deglitched signal shows the quality of the processing.



The wavelet coefficients are crucial for this deglitching process.



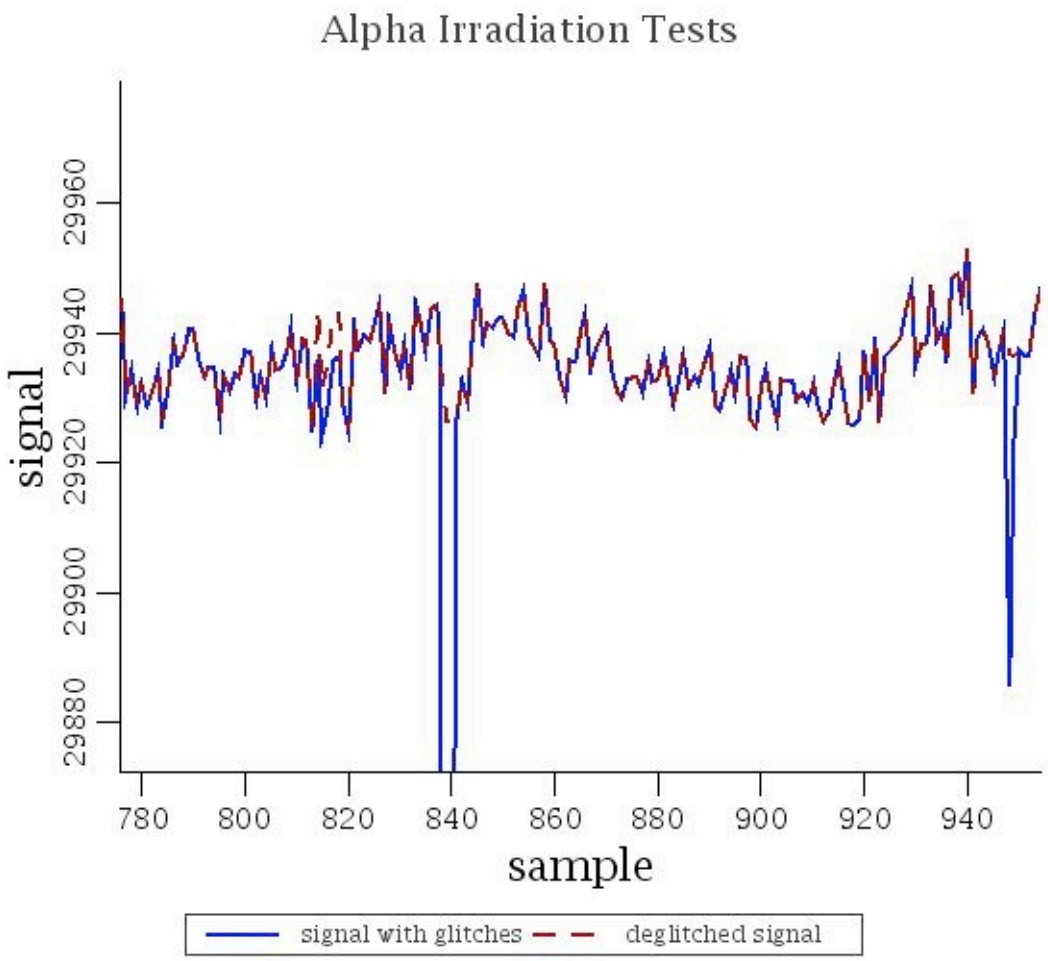
From the image it is obvious, that the glitches are placed according to their width into the wavelet coefficients. This fact is due to the choice of the median transform and the configuration of the environment (plus/minus 1 for coeff 1, plus/minus 2 for coeff 2 etc.). Please note also that coefficient 6 does not contain data (glitches have widths up to 5). Please find details of the Multiresolution Median Transform in Starck et al. (1999). The baseline of the coefficients is zero, not the signal level. The signal level appears only in coefficient 0! Coefficient 0 has also been used to remove the background from the image while the noise has been estimated. From this plot it is obvious, why coeff(0) has been used. Here is a closer look at the wavelet coeficients:



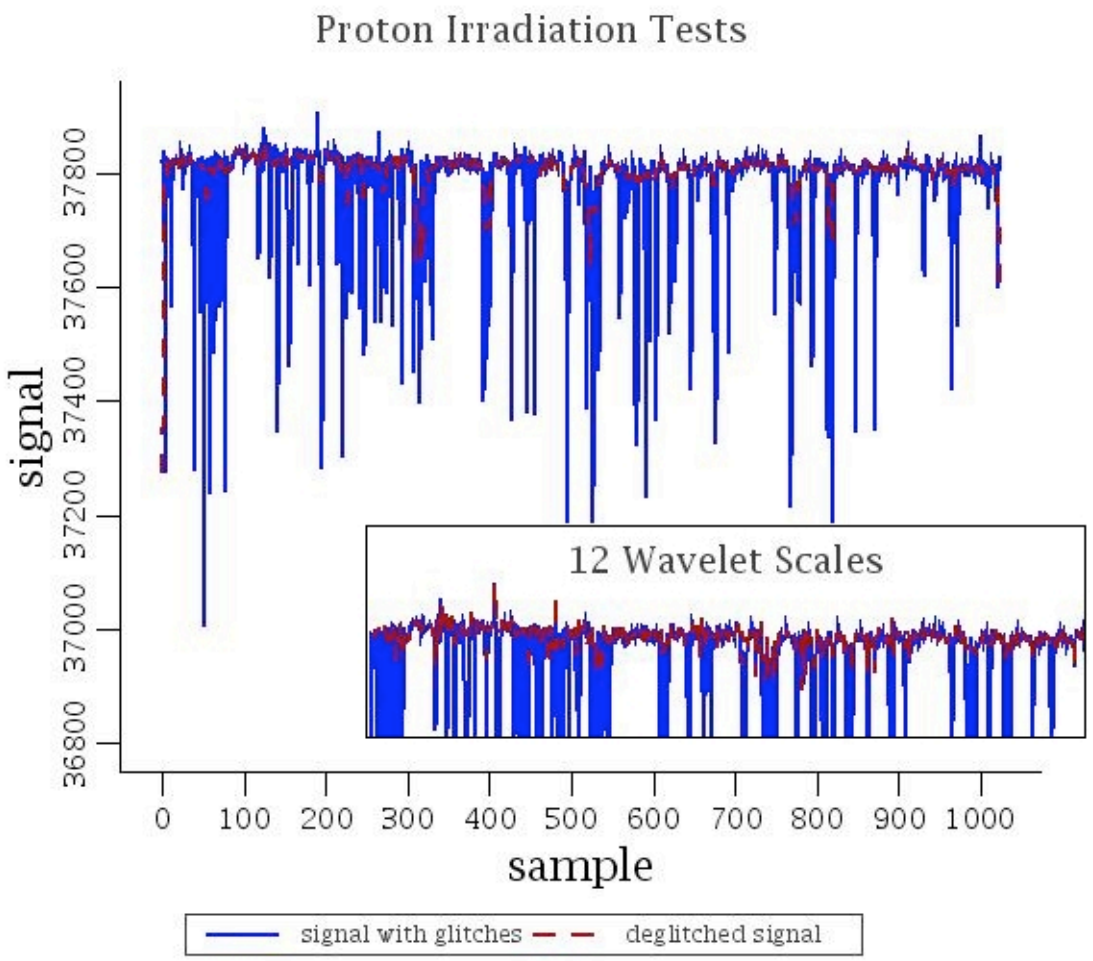
Although the wavelets baseline is zero a little noise is there. Its even in the same order of magnitude as the original Signals noise. Thats why it is important to have a good noise estimate to remove the glitches.

7.8.1.3 Alpha and Proton Irradiation Tests

Here are the results of the Multiresolution Median Transform deglitching applied to the Bolometer irradiation test with alpha particles and protons from (CITATION NEEDED).



The glitches of the alpha particles do not differ significantly from the artificial glitches. Their width is around 1-3 readouts and their signal is much higher than the average signal, so the nsigma method works as well as expected. The proton tests have a distinctly different pattern. The glitches are much higher in number and their width can also be large. The image shows, that increasing the wavelet scales from 6 to 12 leads to a better removal of the glitch structures. The problem in these measurements is that the number of the glitches is so high, that a good estimate of the noise is hard to do. There is no pixel without a glitch. Thus the estimated noise will be high and small glitches are not removed. Anyhow, experimenting with the settings is in every case worth a try as the 12 wavelet scale inset shows.



7.8.2 Wavelet Transform Modulus Maxima Lines Analysis (photWTMMLDeglitching, testing phase)

Thanks to previous missions, we have now some models of the cosmic particles into our considered space area. Despite of these models and the knowledge of the behavior of our detector, it isn't possible to know exactly the glitch signatures that will be encountered in the space, hence it is important to have a flexible deglitching algorithm. The MMT method showed above is well adapted because there is no assumption about the glitch signature. Nevertheless, one can try to work on the shape of the glitch. This direction has been explored by the Spire's developers with the 'WTMML' analysis (Wavelet Transform Modulus Maxima Lines). This method tries to recognize the temporal shape of the glitches. With their courtesy permission, their algorithm has been adapted for PACS. This section tries to give an evaluation of the algorithm with its strength, its weakness and the limits of its applicability. However, we point out that this method is not yet available in the pipeline and it is still in the testing phase. Overview Status : first version - not ready yet. Author b.Marin based on WTMML software developed by C.Ordenovic, C.Surace, B.Torresani, A.Llebaria Reference literature for this algorithm and data used:

- Faint source detection in ISOCAM images, J.L. Starck, H. Aussel, D. Elbaz, D. Fadda, and C. Cesarsky, A&A Suppl. Ser. 138,365-379 (1999)
- Glitches detection and signal reconstruction using wavelet analysis, C.Ordenovic, C.Surace, B.Torresani, A.Llebaria STAMET-D-07-00048
- A wavelet tour of signal processing, Mallat
- Glitch effects in ISOCAM detectors A.Claret, H.Dzitko, J.Engelmann and J.-L. Starck
- Herschel/PACS Description des irradiations Tandem 2005 [Ref: SAp-PACS-BH-0470-05 ver 1.1]

Benoit HOREAU

The concept of the method is the following: for each pixel(x,y), the method consists of doing the following steps :

1. set s(t) = D(x,y,t) ; where D represents sampling data and t the time
2. A multi-resolution decomposition of the signal is done using Mexican Hat wavelet. the result is Ws(b,a)
3. Signal irregularities are tagged by the study of the evolution of |Ws(b,a)| in the time-scale plane(b,a)
4. noise is estimated
5. Irregularities not identified as noise are tagged as glitches
6. Glitch contributions are estimated and removed from the decomposed signal
7. Signal is rebuilt

Details and results of the implementation

```
>> outFrames = photWTMMLDeglitching(inFrames [,copy=copy][,scaleMin=2.0][,scaleMax=6.0]
               [,hmin=-1.3][,voices=5.][,holderThreshold=-0.6][,CorrThreshold=0.985]
               [,reconstruction=True]

inFrames  : the input frame object containing signal to analyse
outFrames : the returned Frame object containing signal deglitched and a mask of pixels
            modified
copy      : boolean with the possible values :
                false (jython: 0) - inFrames will be modified and returned
                true (jython: 1)  - a copy of inFrames will be returned
scaleMin  : signal continuous wavelet transform is computed from scaleMin
scaleMax  : signal continuous wavelet transform is computed till scaleMax
voices    : voices number by octave
hmin      : the minimal holder value allowed
holderThreshold : must be greater than hmin - the threshold holder exponent
corrThreshold   : correlation coefficient threshold (around 1.) is a criteria used
                  to identify an irregularity of the signal
reconstuction   : Boolean: {true|false} = {inFrames is changed | inFrames is not changed}
```

More parameter descriptions Wavelet transform will be computed from scaleMin to scaleMax. Octave number (nOct) is log(scaleMax)/log (2) (dyadic decomposition) and there are nVoice voices by octave. The scale a of the octave o and the voice v is a = 2^(nOct*(o-1)+v/nVoice) Correlation threshold (close to value 1.) is a criteria used to identify a potentially irregularity of the signal as a possible glitch. holderThreshold gives an upper limit of the acceptable holder exponent found hmin gives a lower limit of the acceptable holder exponent found hmin < holder exponent found < holderThreshold Algorithm description

1. Multiresolution signal decomposition is performed from minScale to maxScale. The Mexican hat wavelet is used here.
2. Along each scale, locally maxima are identified. In other words, if dWs(b0,a0)/db = 0 the point b0,a0 is a locally maximum.
3. Across the scales, 'maxima lines' are researched. A maxima line is any curve a(b) in the scale-space plane (b,a) along which all points (previously identified) are modulus maxima.
4. Singularities are detected by finding the abscissa where the wavelet modulus maxima converges at fine scale
    1. glitch signature can be characterized as a Lipschitz function (HÃ¶lder). HÃ¶lder exponent is evaluated thanks to the Mallat inequalities: log2[|Ws(b,a)|]<= log2 (A) + Î± log2(a) for a -> 0 ; Î± is the local HÃ¶lder exponent ; Ws(b,a) is the wavelet coefficient of the signal s at the scale a and the time b along the maxima line
    2. an irregularity of the signal can produce a cone in scale/time referential, so the coefficient correlation C is calculated on the set of points (log2(|Ws(bi,ai)|),log2 (ai))
    3. when C is greater than our corrThreshold, the linear regression is performed between minScale and maxScale and the slope of the linear regression can give a holder exponent

4. if the holder exponent found is between hmin and holderThreshold, a singularity/detection has been found, and we know the contribution of the cone on the signal through wavelet coefficients

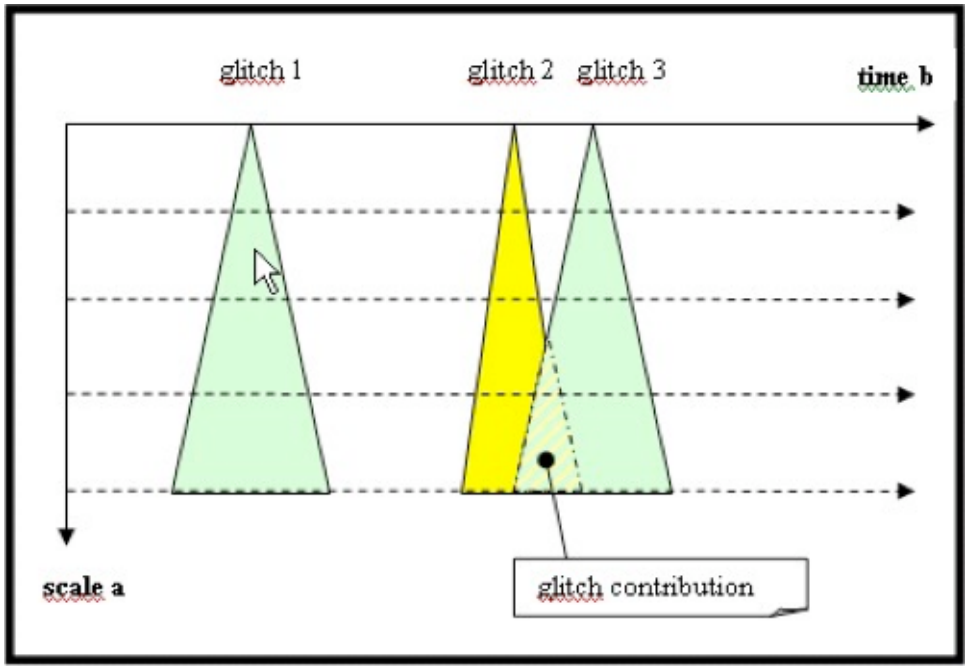
5. false detections provided by the noise are identified and removed by sigma clipping algorithm applied to the wavelet coefficients
    1. white noise is a stationary process having the same spectral density whatever the frequencies. Thanks to Donoho, one can compute from the lowest scale (a=1) and wavelet coefficients found at this scale, an estimator of the noise variance :

$$\sigma^2 = \left(med\left(\frac{|W_s(b,a=1)|}{0.6745}\right)\right)^2$$

    2. From the detections found by the maxima line analysis, one can considered white noise contribution when the wavelet coefficient are lower than 3sigma

$$|W_s(b,a)| < 3\sigma$$

6. Glitch wavelet coefficient contributions are calculated and removed from the signal

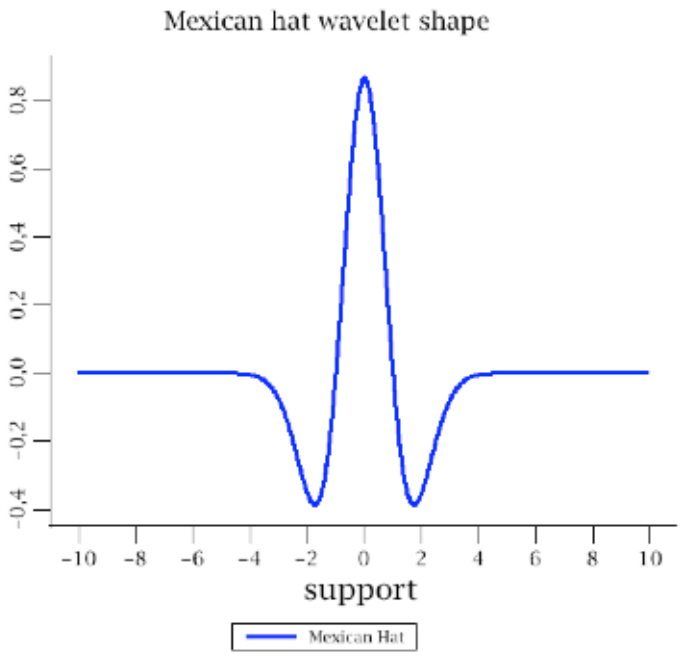


7. Signal is rebuilt with the following synthesis equation

Wgl : glitch contribution, Ws : wavelet coefficients of the signal

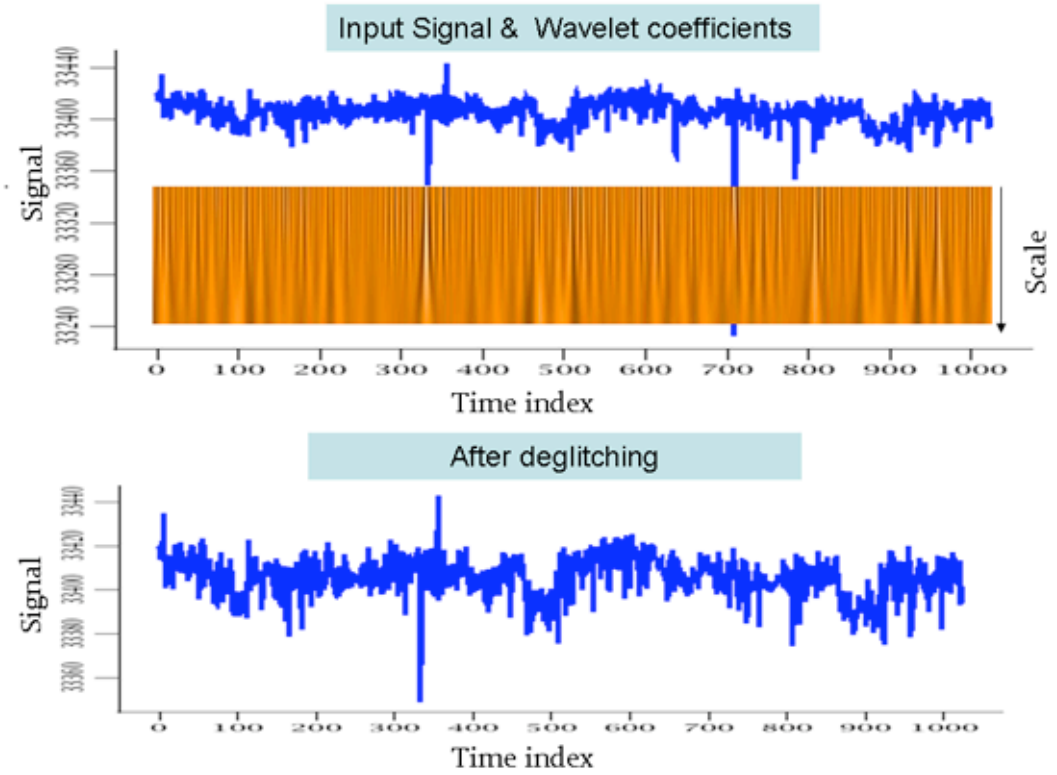
$$s(t) = \frac{1}{C_\psi} \sum_i \left[W_s(t,a_i) - W_{gl}(t,a_i)\right]\frac{da_i}{a_i}$$

Wavelet used:



Results:

The following figure gives an overview of the deglitching process. wtmmlDeglitching task has been run with the default parameter. At the top, one can see the input signal extracted from the real data got during the alpha irradiation tests. In the middle (in orange), the wavelet coefficients according to the scale are plotted. At each signal variation, one can see a cone. This cone is analyzed, glitches are identified, removed, then the signal is rebuilt. At the bottom, the signal has been deglitched. On the right, the glitch located at position 710 has been removed, while the glitch at position 332 is still there. Beware that y axis at the top and at the bottom, are not identical.



Conclusion

The tuning of the parameters of this software is tricky. One's want to remove efficiently the glitches encountered, one can run several times the WTMML algorithm with various parameters.

Application domain:

If the deglitching of the data delivered by the MMT method is considered as insufficient, one can use wtmmlDeglitching task with various parameters. This task will be useful for a limited usage, ran individually according to the glitch signature.

```
>> success = wtmmlDeglitchingTest([signalLevel=1e-12][,signalShape=0]
                          [,noise="gaussian"][,noiseLevel=5.][,glitchNumber=3]
                          [,glitchLevel=8.] [,glitchShape="lipschitz"))

success : boolean: {true|false} = {successful tests|tests have failed}

Optional parameters are only for Test5 :
signalLevel : whatever value
signalShape : {0,4}
noise : {"gaussian","white"}
noiseLevel : sigma value from the signal
glitchNumber : number of glitch generated
glitchLevel : sigma value from the signal (must be higher than the noiseLevel)
glitchShape : {"lipschitz","amortizedSine"}↵
```

7.9 addUTC (not implemented yet)

Convert from spacecraft on-board time (OBT) to UTC, using the time correlation table. Fill the UTC field in the frames dataset.

```
>> outFrames = addUTC(inFrames frames, timecor,[,copy=copy][,calVersion = calVersion])
```

```
outFrames  : Frames       : Frames out
inFrames   : Frames       : Frames in
timecor    : TableDataset : Time corrections
copy       : int          :
                           0 - return reference
                           1 - return copy
calVersion : String : Version of the calibration files used
```

Calibration File: Time correlation table

7.10 convChopper2Angle (jython prototype available)

This task converts the Chopper position expressed in technical units to angles. This is done by reading the CPR entry in the Status table and express it in two ways: a) as angle with respect to the FPU (CHOPFPUANGLE entry in the Status table) and b) as angle in the sky (CHOPSKYANGLE). Both angle are in arcseconds. In particular, the CHOPFPUANGLE is a mandatory input for the PhotAssignaRaDec task, to be executed after Level 0.5 for the final step of the astrometric calibration. Thus, the convChopper2Angle task must be executed even if the chopper is not used at all as in the scan map (chopper maintained at the optical zero). CHOPFPUANGLE corresponds to the chopper throw in arcseconds in HSpot.

```
>> outFrames = convertChopper2Angle(inFrames [,copy=copy]  [, calVersion = calVersion])

outFrames  : Frames : Frames out
inFrames   : Frames : Frames in
copy       : int    :
                     0 - return reference
                     1 - return copy
calVersion : String : Version of the calibration files used
```

The calibration between chopper position in technical units (voltages) and angles is give by a 6th oder polynomial. The calibration is based on the calibration file containing the Zeiss conversion table. Reference: "Angular Calibration and zero-point offset determination of PACS FS Chopper for cold HeII (T=4.2 K) conditions.", PICC-MA-TR-009, U. Klaas, J. Screiber, M. Nielbock, H. Dannerbauer, J. Bouwman.

7.11 convXYStage2Pointing (available)

During the so-called PACS ILT tests in the lab, there was no info about satellite pointing information. So this step is used to simulate pointing information for this particular test-case. For real PACS Herschel data the next task, "photAddInstantPointing", should be used instead. The coordinates of the used point source, called XY stage, are included in the Status table and used later as input for a simulated astrometric calibration (photAssignRaDec).

```
>> outFrames = convXYStage2Pointing(inFrames, seq [, noInter=noInter] [,copy=copy])

outFrames : Frames         : Frames out
inFrames  : Frames         : Frames in
seq       : PacketSequence : PacketSequence holding the TmPackets of the period of
                             Frames
noInter   : boolean        : True  : without Interpolation
                             False : with Interpolation (default !)
copy      : int    :
                     0 - return reference
                     1 - return copy
```

The coordinates of the XY stage are contained in the XY HK. This info is extracted from there and the internal time is used to merge the coordinates to the individual frames. The HK packets have a readout frequency lower than the frames readout. So the task by default (noInter = false) interpolates between the available XY stage coordinates to obtain coordinates for each frame. With the keyword "noInter=true" no interpolation is done.

New entries in the Frames Status :

1. XY_Stage_EvType : Event Type (regular, start , stop)
2. XY_Stage_Mode : Mode (idle ,local single, local raster, single position move, single raster)
3. XY_Stage_TimeSec : Time seconds
4. XY_Stage_TimemS : Time miliSeconds
5. XY_Stage_LV_Sts :
6. XY_Stage_Status : XY Stage Status
7. XY_Stage_X_Axis : X axis position
8. XY_Stage_Y_Axis : Y Axis position
9. XY_Stage_X_idx
10. XY_Stage_Y_idx
11. XY_Stage_Stage_Nod_cnt : Nodding count
12. XY_Stage_Nod_pos : Nodding position (on raster , off raster)
13. XY_Stage_column : Column count
14. XY_Stage_line : Line count

This task allows also to include info about the nod cycle by adding a nod position counter (entry XY_Stage_Stage_Nod_cnt) and the a nod on or off position identifier (entry XY_Stage_Nod_pos).

7.12 photAddInstantPointing

The purpose of this task is to perform the first step of the astrometric calibration by adding the sky coordinates of the virtual aperture (center of the bolometer) and the position angle to each readout as entry in the status table. In addition the task associates to each readout raster point counter and nod counter for chopped observations and sky line scan counter for scan map observations.

```
>> outFrames = photAddInstantPointing(inFrames, scPointing [,copy=copy])

outFrames : Frames : Frames out
inFrames  : Frames : Frames in
scPointing : PointingProduct : Pointing information
copy      : int    :
                     0 - return reference
                     1 - return copy
```

This first part of the astrometric calibration deals with two elements: the satellite pointing product and the SIAM product. Both are auxiliary products of the observation and are contained in the Observation context delivered to the user. The satellite pointing product gives info about the Herschel pointing. The SIAM product contains the a matrix which provides the position of the PACS bolometer virtual aperture with respect to the spacecraft pointing. In the current version of the pipeline this task used a SIAM matrix contained in a calibration file and not the one of the SIAM product. However, this will be changed in the future and the SIAM product will be used for the astrometric calibration. The time is used to merge the pointing information to the individual frames. The task adds the following entries to the status table:

1. - RaArray: ra coordinate of the virtual aperture (deg)
2. - DecArray: dec coordinate of the virtual aperture (deg)
3. - PaArray: position angle (deg)
4. - raArrayErr: ra coordinate inaccuracy of the virtual aperture (deg)
5. - decArrayErr: dec coordinate inaccuracy of the virtual aperture (deg)
6. - PaArrayErr: position angle inaccuracy(deg)
7. - Mode: PacsPhoto, in the bolometer case
8. - RasterLineNumber, for chooped observation only
9. - RasterColumnnumber, for chooped observation only
10. - NodcycleNum, for chopped observation only

11. - OnTarget, on source position identifier, for chooped observation only (flase or true)
12. - AbPosId, for chooped observation only (false or true)
13. - IsSlew, identifies satellite slewing (false or true)
14. - IsOffPos, identifies off position (false or true)
15. - ScanLineNumber, identifies the scan line number for scan map observation
16. - AcmcMode
17. - Aperture
18. - IsAposition, identifies A position in nod cycle
19. - IsBPosition, identifies B position in nod cycle
20. - IsOutOfField
21. - IsSerendipity
22. - RollArray, any difference with PAArray???

7.13 cleanPlateau (java prototype available)

This task is executed before Level 0.5 only for chopped observations (point-source, small-source, chopped raster modes).

```
>> outFrames = cleanPlateauFrames(Frames inFrames[,dmcHead=dmcHead][,copy=copy]
                    [,calVersion = calVersion])

outFrames  : Frames : Frames out with mask UNCLEANCHOP
inFrames   : Frames : Frames in
copy       : int    :
                     0 - return reference
                     1 - return copy
calVersion : String : Version of the calibration files used
```

The module flags the readouts at the beginning of a chopper plateau, if they correspond to the transition between two chopper positions. In the chopper transition phase, the chopper is still moving towards to proper position and the signal of this readouts does not correspond to the on or off position. Usually the chopper is moving so fast that only one readout needs to be masked out. The module just adds the 3D UNCLEANCHOP mask to the input frame. The task identifies the chopper plateaus on the basis of the CHOPPERPLATEAU (for the science data) and CALSOURCE (for the calibration block) entries in the status table. For each chopper plateau the readouts with a chopper position deviating from the mean position (threshold provided by the calibration file ChopJitterThreshold) are flagged in the UNCLEANCHOP mask.

8 The AOT dependent pipelines

After level 0.5, the pipeline is AOT dependent. In the following sections we will describe separately the different AOT pipelines, point source, small source, chopped raster, scan map AOTs, up to Level 2.

9 Point Source AOR

9.1 Level 0.5 to Level 1

9.1.1 photMakeDithPos (jython prototype available)

The task just checks if exists a dithering pattern and identifies the dither positions. The task adds a dither position counter, "DithPos", to the Status table. Frames with the same value of 'DithPos' are at the same dither position.

```
>> outFrames = photMakeDithPos(inFrames [,copy=copy] )

outFrames  : Frames : Frames out with one image per every single chopper plateau
inFrames   : Frames : Frames in
copy       : int    : This has to be done by
                     0 - return reference
```

```
                    1 - return copy
```

9.1.2 photMakeRasPosCount (jython prototype available)

The task adds raster position counter to status table.

```
>> outFrames = photMakeRasPosCount(inFrames [,copy=copy])

outFrames  : Frames : Frames out with one image per every single chopper plateau
inFrames   : Frames : Frames in
copy       : int    : This has to be done by
                      0 - return reference
                      1 - return copy
```

The task needs the output of the photAddInstantPointing task to be executed otherwise an error is raised saying that the pointing information are missing for the observation. The module uses the virtual aperture coordinates and the raster flags in the status table to identify different raster positions. The raster positions are identified in the Status table by the new entries 'OnRasterPosCount' and 'OffRasterPosCount'.

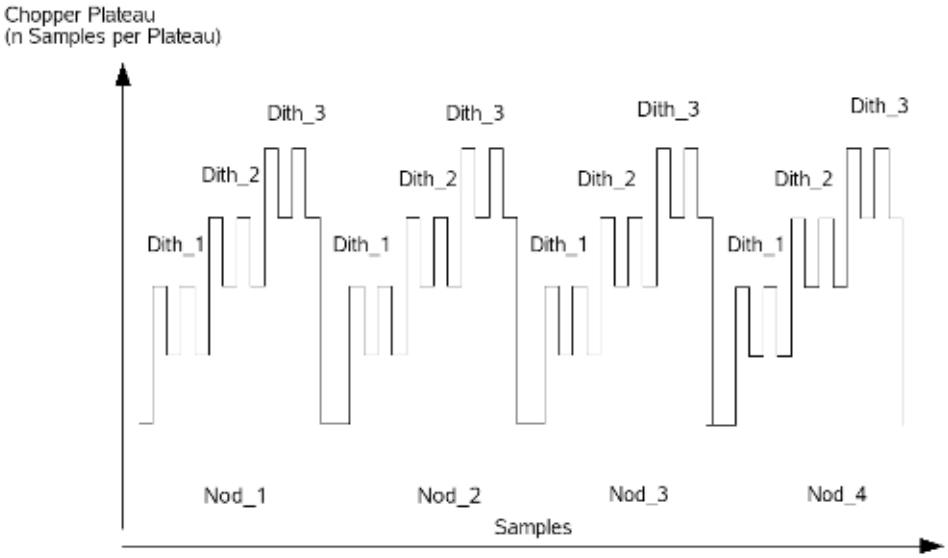
9.1.3 photAvgPlateau (java prototype available)

The task averages all valid signals on chopper plateau and resamples signals, status and mask words for the photometer. It calculate noise map but not the coverage map. The result is a Frames class with one image per every single chopper plateau.

```
>> outFrames = photAvgPlateau(inFrames [,sigclip=0] [,mean=0]
                     [,qualityContext=QualitxContext]  [,copy=0] )

outFrames       : Frames : Frames out with one image per every single chopper plateau
inFrames        : Frames : Frames in
sigclip         : Value for sigma clipping (default = 0 : no sigma clipping)
mean            : mean = 1 : use MEDIAN, mean=0: use AVERAGE
qualityContext  : QualityContext
copy            : int    : This has to be done by
                     0 - return reference
                     1 - return copy
```

The module uses the status entry CHOPPERPLATEAU (CALSOURCE in case of calibration block pre-processing) to identify the chopper plateau in the same way as CleanPlateau. Then it computes the average (sigma clipping if sigclip > 0, and median if mean =1) for each pixel over the chopper plateau .

Simplified Example : Chopper Plateaus



The signal of the bad pixels, identified by the BADPIXEL mask, is reduced by the task as the unmasked pixel. The pixels flagged in the other available masks (SATURATION, GLITCH, UNCLEANCHOP) are discarded in the average. If the chopper plateau contains no valid data (all pixels masked out) the signal is set to NaN (Not a Number). The noise is calculated for each pixel (x,y) and each plateau (p) as:

noise[x,y,p] = STDDEV(signal[x,y,validSelection[p]]) / SQRT(nn)

where nn is the number of valid readouts in the chopper plateau. This number is then stored as addition entry (NrChopperPlateau) in the status table. The noise is stored in the Noisemap

The Status entries with different values over the chopper plateau length are modified with the following scheme:

- OBSID: value of the beginning of the chopper plateau
- BBID: value of the beginning of the chopper plateau
- LBL : removed
- TMP1: removed
- TMP2: removed
- FINETIME: value of the beginning of the chopper plateau
- VLD : removed
- WPR: value of the beginning of the chopper plateau
- BOLST: removed
- BSID : removed
- CRDC: value of the beginning of the chopper plateau
- CRDCCP: value of the beginning of the chopper plateau
- DBID: value of the beginning of the chopper plateau
- DMCSEQACTIVE: value of the beginning of the chopper plateau
- CHOPPERPLATEAU : Sum
- CALSOURCE : Sum
- PIX: removed
- RCX: removed
- RESETCNT: Just counting 1 to x
- BLOCKIDX: removed
- BAND: value of the beginning of the chopper plateau
- BBTYPE: value of the beginning of the chopper plateau

- BBSEQCNT: value of the beginning of the chopper plateau
- UnCleanChop: Sum
- DithPos : Median
- OnRasterCount : Median
- OffRasterCount : Median

9.1.4 photDiffChop (java prototype available)

Subtract every off-source signal from every consecutive on-source signal. The result is a Frames class with one image per one chopper cycle.

```
>> outFrames = photDiffChop(inFrames ,hkdata=hkdata [,qualityContext=QualityContext]
                     [,copy=0] )

outFrames          : Frames : Frames out with one image per one chopper cycle
inFrames           : Frames : Frames in
hkdata             : TableDataset : issued from HPPHK product
                     (Herschel PACS Photometer HK)
qualityContext     : QualityContext
copy               : int    :
                         0 - return reference
                         1 - return copy
```

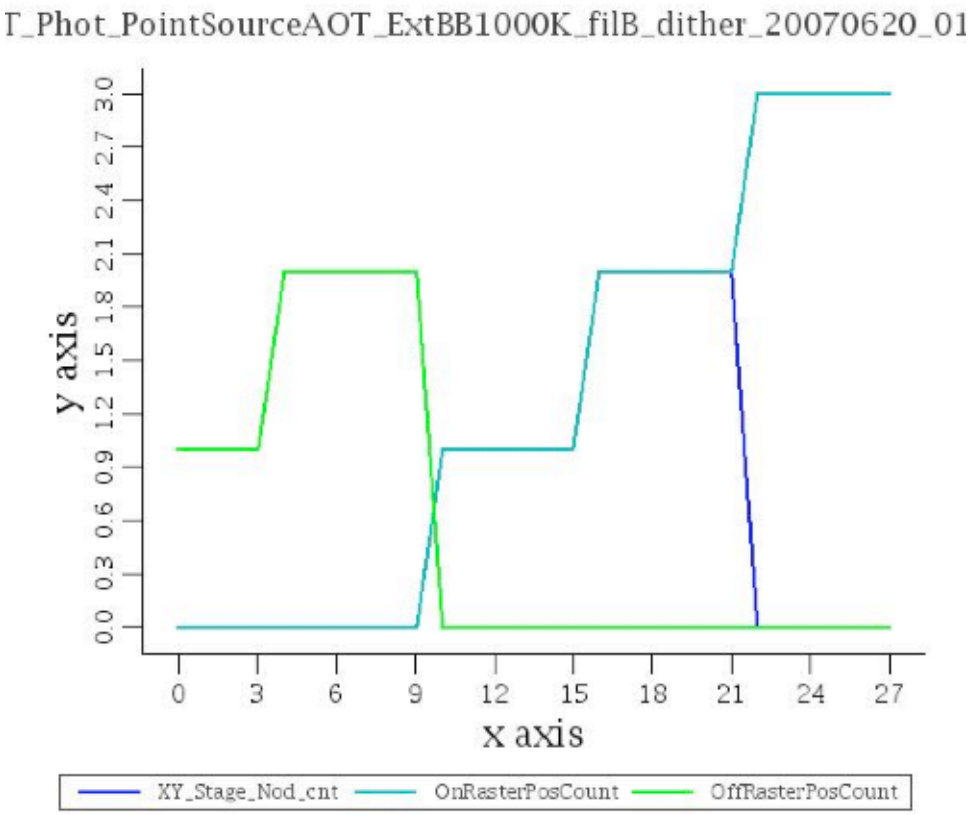
To better subtract the telescope background emission and the sky background the 'off-source' image is subtracted from the 'on-source' image (consecutive chopper positions). The module accepts as input the output of photAvgPlateau module. It returns as output a Frames class with the differential image of any couple of on-off chopped images. The module resamples the status table and the the masks accordingly (see NOTE in photAvgPlateau section).

The on and off images are identified on the basis of the status entries added by the photAddInstantPointing task. The noisemap is computed in the following way:

noise [x,y,k] = SQRT(noise[x,y,pON]**2 + noise[x,y,pOFF]**2)

where k is the frame number of the differential on-off image, pOn is the frame number of the on source image, pOFF is the frame number of the off source image, noise[x,y,pON] and noise[x,y,pOFF] are the error maps at the on and off source images, respectively (output of the previous pipeline step).

Simplified Example : Chopper Plateaus



9.1.5 photAvgDith (jython prototype available)

The chop cycle is repeated several times per any A and B nod position. This task calculates the mean of the on-off differential chopped images per any A and B position within any Nod cycle. If the dithering is applied in the point-source mode as offered by HSpot, the average is done separately per dithered A and B nod positions.

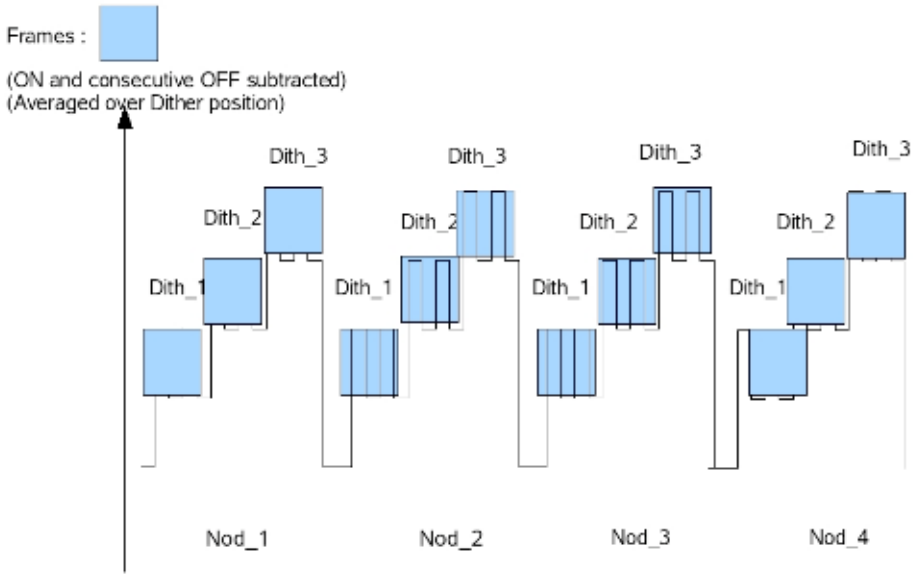
```
>> outFrames = photAvgDith(inFrames [,qualityContext=QualityContext] [,copy=0] )

outFrames      : Frames : Frames out with one image per chopper plateau per
                 nodding position
inFrames       : Frames : Frames in
qualityContext : QualityContext
copy           : int    :
                       0 - return reference
                       1 - return copy
```

The task uses several entries in the status table to identify the on-off differential images (output of photDiffChop) belonging to the A and B Nod position of a given Nod cycle and dithered position (DithPos, NodcycleNum, IsAPosition, IsBPosition, see output of photAddIstantPointing). Since only the average of the identified images is performed, the noise is propagated as follows:

For "c" chopper cycles (c=k), we average the n/2 differences

noise [x,y] = SQRT(MEAN(noise[x,y,:]**2)) / SQRT(n)

Simplified Example : Chopper Plateaus



9.1.6 photDiffNod (java prototype available)

This task is performing the last step of the background (sky+telescope) subtraction. It subtracts the images corresponding to the A and B positions of each nod cycle and per each dither position. The module needs as input the output of photAvgDith.

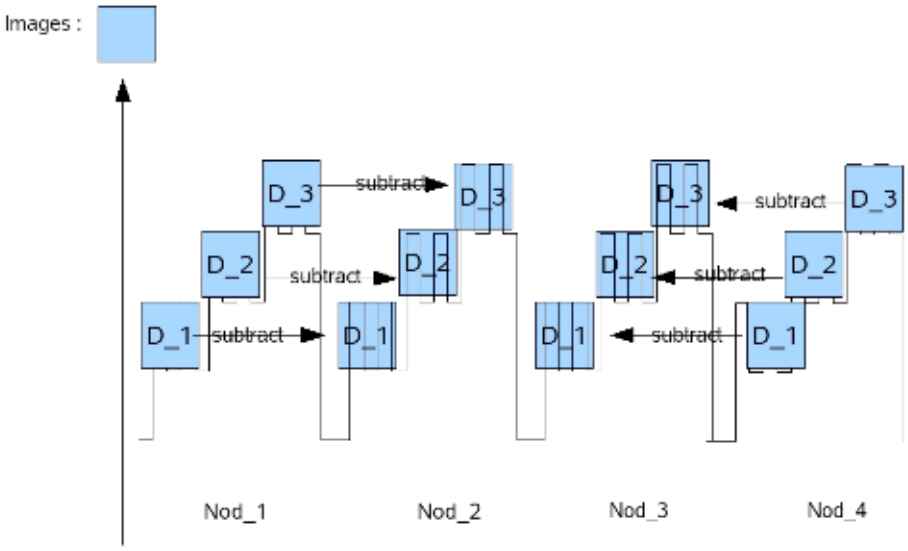
```
>> outFrames = photDiffNod(inFrames [qualityContext=QualityContext] [,copy=0] )

outFrames      : Frames  : Frames out with one image per nod cycle
inFrames       : Frames  : Frames in
qualityContext : QualityContext
copy           : int     :
                       0 - return reference
                       1 - return copy
```

The noise is propagated as follows:

noise [x,y,k] = SQRT(noise[x,y,A]**2 + noise[x,y,B]*+2)

where the A and B indexes refer to the A and B nod position.

Simplified Example : Chopper Plateaus



9.1.7 photCombineNod (java prototype available)

The nod cycles are repeated many times per any dither position. This task is taking the average of the differential noda-nob images corresponding to any dither position. The results is a frames class containing a completely background subtracted point source image per any dither position.

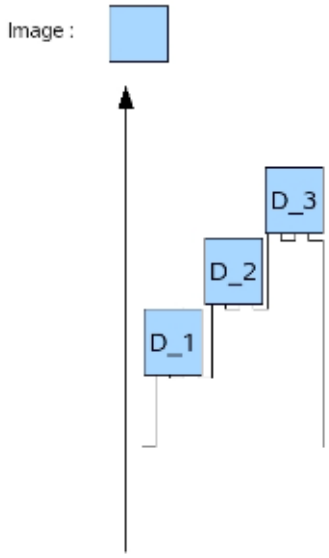
```
>> outFrames = photCombineNod(inFrames [qualityContext=QualityContext] [copy=0] )

outFrames      : Frames  : Frames out with one image
inFrames       : Frames  : Frames in
qualityContext : QualityContext
copy           : int     :
                     0 - return reference : overwrites the input frames (default)
                     1 - return copy
```

The noise is propagated as follows:

noise[x,y,d] = STDDEV(signal[x,y,nd]) /SQRT(nd)

where d is the index of the dither position and nd is the number of nod cycles per dither position.

Simplified Example : Chopper Plateaus



9.2 Level 1 to Level 2

9.2.1 photDriftCorrection

The task applies the drift correction of the flat field and controls the photometric stability

```
>> outFrames = photDriftCorrection(inFrames [,calTree=calTree][,dCSsRef=dCSsRef]
                        [threshold=threshold][algo=algo][,copy=copy] )

outFrames    : Frames              : Frames out in Jansky photometricaly calibrated
inFrames     : Frames              : Frames with signal in Volt
calTree      : PacsCalibrationTree : calibration tree containing all calibration
               products used by the pipeline
dCSsRef      : DCSRef              : DC0 reference differential image CS1-CS2
               (difference of the two internal calibation sources, CS1 and CS2);
               by default it is taken from the calibration file
threshold    : PhotometricStabilityThreshold
algo         : String              : algorithm used when many calibration blocks exists
               {"first","last","mean" [default] ,"median","interpolation"}
               "first"  - only the first calibration blocks is used
               "last"   - only the last calibration blocks is used
               "mean"   - the mean of calibration blocks is used
               "median" - the median of the calibration blocks is used
               "interpolation" - an interpolation is done between the calibration blocks
copy         : int                 :
                                   0 - return reference
                                   1 - return copy
```

Literature

Photometric calibration of PACS bolometer - K.Okumura, D.Lutz, M.Sauvage and B.Morin - October 11,2007

Photometric calibration products - B.Morin, K.Okumura, D.Lutz, M.Sauvage - February 08,2008

The PhotDriftCorrection task has the goal to multiply signal s(t) by the ratio DC0/DCs, where DC0 is the differential image of the two internal calibration sources (calculated from the same data of the flat-field), DCs is the differential image of CS1-CS2 obtained from the calibration block of the observation (output of the cal-block pre-processing). This factor corrects possible drift of the flat field. This drift can be due either to an alteration of the internal calibration sources or to an evolution of the detector pixels. The drift is compared with photometric stability threshold parameters (stored in the calibration files). If the ratio

overtakes these thresholds, a 'DriftAlert' keyword is added to the metadata .

The formula managing the flat-field, the flux calibration and the photometric adjustement is the following:

$$\delta f(t) = \delta s(t) * \frac{(\delta C_0)}{(\delta C)} * \frac{1}{(J\Phi)}$$

where f(t) is the flux in Jy, s(t) is the signal in Volt, DC_0 is the difference of the calibration sources got during a calibration campaign, DCs is the difference of the calibration sources computed by the cal-block pre-processing, J is a flux calibration factor which contains the responsivity and the conversion factor to Jansky, Phi is the normalized flatfield. The ratio 1/J*Phi converts the signal s(t) in Volt to f(t) in Jansky. THE Ratio DC_0/DCS corrects the drift of the flat field and flux calibration.

Hereafter the formula used to compute the noise

$$^{*}noise = SQRT(s_{out}^{2} * [(sigmas^2/s^2) + (sigmaC_0^2/C_0^2) + (sigmaC_s^2/C_s^2)])$$

where s is the input signal in Volt, sigmas is the input noise, C0 is our reference, $sigmaC_0$ is the noise of the reference, DCs is the diferential image of the cal-block and signaDCs is the noise associated to that.

Addendum: the first DC_0 has been determined with data collected during ILT test campaign. The following biases have been used: 2.6 V for both the blue and green channel, 2.0 V for the red one.

9.2.2 photRespFlatFieldCorrection

The task applies flat field corrections and converts signal to a flux density

```
>> outFrames = photRespFlatFieldCorrection(inFrames [,calTree=calTree]
                    [,flatField=flatField]
                    [,responsivity=responsivity][,copy=copy] )

outFrames    : Frames              : Frames out in Jansky photometricaly calibrated
inFrames     : Frames              : Frames with signal in Volt
calTree      : PacsCalibrationTree : calibration tree containing all calibration products
               used by the pipeline
flatField    : FlatField           : FlatField calibration product
responsivity : Responsivity        : Calibration product converting the signal in
               Flux density (Jansky)
copy         : int                 :
                                   0 - return reference
                                   1 - return copy
```

The formula managing the flat-field, the flux calibration and the photometric adjustement is the following:

$$\delta f(t) = \delta s(t) * \frac{(\delta C_0)}{(\delta C)} * \frac{1}{(J\Phi)}$$

where f(t) is the flux in Jy, s(t) is the signal in Volt, DC_0 is the difference of the calibration sources got during a calibration campaign, DCs is the difference of the calibration sources computed by the cal-block pre-processing, J is a flux calibration factor which contains the responsivity and the conversion factor to Jansky, Phi is the normalized flatfield. The ratio 1/J*Phi converts the signal s(t) in Volt to f(t) in Jansky. This task applies the ratio 1/J*Phi to flat-filed and flux calibrate the data.

The noise is calculate in the following way:

$$\frac{\sigma^2 f}{f^2} = \frac{\sigma^2 s}{s^2} + \frac{\sigma^2 \delta C_0}{\delta C^2_0} + \frac{\sigma^2 \delta C}{\delta C^2} + \frac{\sigma^2 K}{K^2} + \frac{\sigma^2 r}{r^2} + \frac{\sigma^2 \phi}{\phi^2}$$

9.2.3 photShiftDith

The dithering pattern offered by HSpot is just a 1/3 pixel shift. Thus the coaddition of the 3 dithered double differential image is done only in pixel coordinates by this task. This is not the definitive result of the pipeline since also for the Point-source mode a final astrometric calibrated image should be provided. This is a work in progress and still under investigation.

10 Small Source AOR

Many of the tasks of this session are the same already described in the Point-Source pipeline. Thus, the description will not be repeated here. For those tasks the user can refer to the previous section.

10.1 Level 0.5 to Level 1

10.1.1 photMakeRasPosCount (jython prototype available)

See description of the same task in the Point-source pipeline

10.1.2 photAvgPlateau (java prototype available)

See description of the same task in the Point-source pipeline

10.1.3 photDiffChop (java prototype available)

See description of the same task in the Point-source pipeline

10.1.4 photAvgNod (jython prototype available)

See description of the same task in the Point-source pipeline

10.1.5 photDiffNod

See description of the same task in the Point-source pipeline

10.1.6 photDriftCorrection (jython prototype available)

See description of the same task in the Point-source pipeline

10.1.7 photRespFlatFieldCorrection (jython prototype available)

See description of the same task in the Point-source pipeline

10.2 Level 1 to Level 2

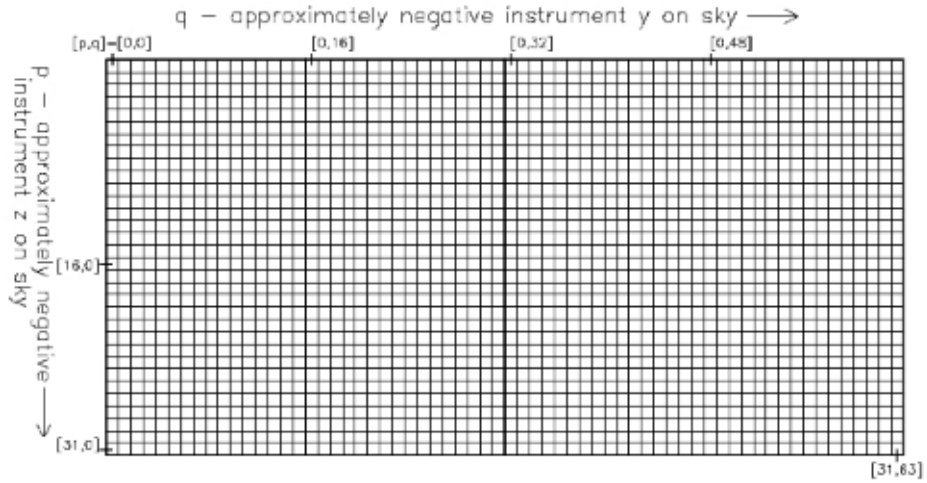
10.2.1 photAssignRaDec

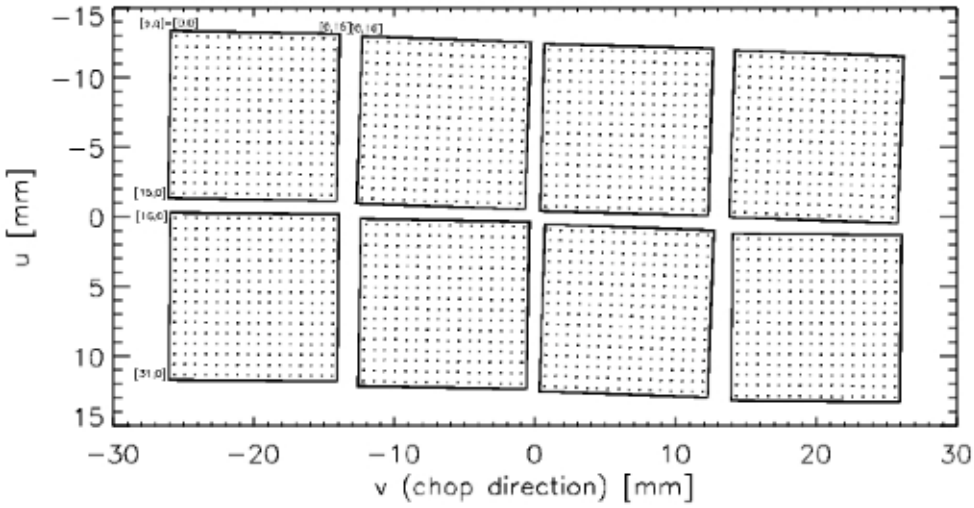
This task performs the last step of the astrometric calibration. Sofar only the sky coordinates of the virtual aperture (center of the bolometer) and the position angle are available in the status table for each frame. the astrometric calibration is done by estimating the sky coordinates of the center of each pixel. This information is then stored into two cubes, one for RA and one for DEC, with the same dimensions of the frame class to be atrometrized.

```
>> outFrames = photAssignaRaDec(inFrames, calTree=calTree, [,copy=0]  )

outFrames          : Frames : Frames out with one image per one chopper cycle
inFrames           : Frames : Frames in
calTree            : PacsCalibrationTree : calibration tree containing all calibration
                     products used by the pipeline
copy               : int    :
                         0 - return reference
```

```
                    1 - return copy
```

This step of the astrometric calibration is done in two steps. In the first step the subarray coordinate system, that is the the integer coordinates (p,q in the figure below) of the pixel centers as displayed in IA, have to be transformed into the the cartesian coordinate system of the PACS focal plane (u, v, respectively, in mm), which reproduces the real misalignment and rotation of the submatrices, as shown in the bottom figure below. The transformation coefficient between p,q to u,v coordinates are contained in the spatial calibration file PCalPhotometer_SubArrayArray_version.fits.



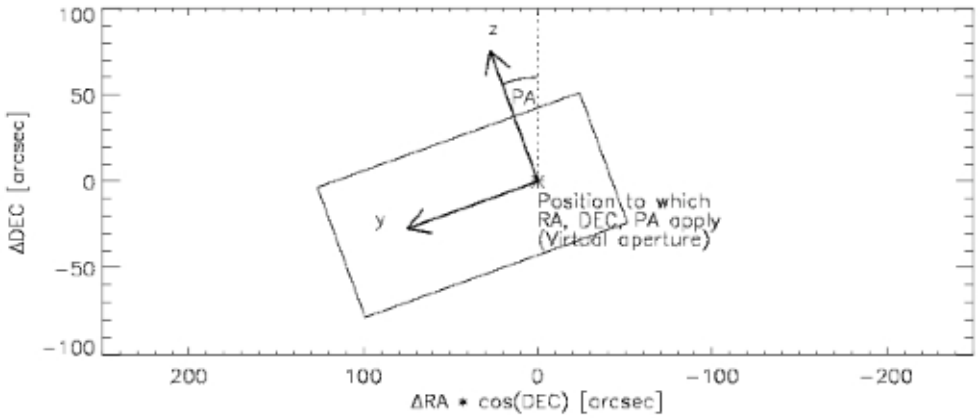


As a second step the coordinates u,v on the PACS focal plane have to be transformed into orthogonal local coordinates on the tangential plane on the sky, y,z. The coordinates y,z, as shown in the figure below, correspond to the offset in arcsecond of the individual pixel coordinates with respect to the the virtual aperture. They are approximated by two polynomials in the three-dimensional space of u,v and chopper and alpha (CHOPFPUANGLE entry in the status table output of convertchopper2angle task):

$$y = \sum_{i=0}^{N} \sum_{j=0}^{M} \sum_{k=0}^{O} a_{ijk} u^i v^j \alpha^k$$

$$z = \sum_{i=0}^{N} \sum_{j=0}^{M} \sum_{k=0}^{O} b_{ijk} u^i v^j \alpha^k$$

the coefficients of the two polynomials are contained in the spatial calibration file PCalPhotometer_ArrayInstrument_version.fits. Once the Instrument coordinates are available, the sky coordinates of the center of each pixel are simply obtained by spherical trigonometric for any given RA and DEC of the virtual aperture and position angle PA (listed for each frames in the status table), as shown in the figure below.

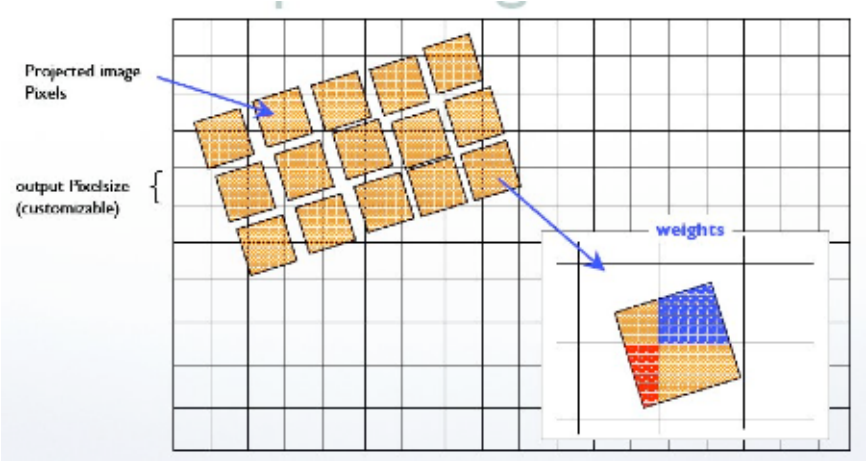


10.2.2 photProject

The protProject task provides one of the two methods adopted for the map creation from a given set of images (in the PACS case, a frame class). The second method is MadMap which will be discussed in the ScanMap pipeline section.

```
>> si = photProject(inFrames, [outputPixelSize=outputPixelsize,] [copy=1,] [monitor=1]
              [optimizeOrientation=optimizeOrientation]
              [mapcoordinates=mapcoordinates] [calibration=calibration])

si              : final map (SimpleImage) with WCS

inFrames        : Frames  : astrometric calibrated input frames

outputPixelSize    : double :  the size of a pixel in the output dataset in arcseconds.
                     Default is the same size as the input (6.4 arcsecs for the red and 3.
                     arcsecs for the blue photometer)
copy            : default is 0 (no copy of inFrames). Option is 1, if inFrames should be c

optimizeOrientation: rotates the map by an angle between 1 and 89 degrees in order to avo
                      huge output maps with lots of zero-signal pixels. Possible vaules: f
                      (default, no automatic rotation), true (automatic rotation)

mapcoordinates: allows to specify the coordinates of the output map. Required values:
                mapcenterra (deg), mapcenterdec (deg), mapwidthra, mapwidthdec, angle.
                If mapcoordinates are given the optimazeOrientation will be ignored even
                if set to true.

monitor         : shows the map montor that allows a close visual inspection of the map bu
                  process. default value is 0 (no map monitor). monitor = 1 shows the map

calibration : default 1 to calculates pixel corners with standard bolometer astrometric
              calibration; 0 for geometrical calculation (test purposes)
```

The task performs a simple coaddition of images, by using a simplified version of the drizzle method (Fruchter and Hook, 2002, PASP, 114, 144). It can be applied to raster and scan map observations without particular restrictions. The only requirement is that the input frame class must be astrometric calibrated, which means, in the PACS case, that it must include the cubes of sky coordinates of the pixel centers. Thus, photAddInstantPointing and photAssignRaDec should be executed before PhotProject. There is not any particular treatment of the signal in terms of noise removal. The 1/f noise is supposed to be removed before the execution of this task, e.g. by the previous steps of the pipeline in the case of chooped-nodded observations and by the photHighPassFilter or similar tasks in the scan map case. The simple projection is shown in the following picture.



First the task defines the dimensions of the output image on the basis of the input images. The size of the output pixel can be specified by the user in arcseconds by setting the outputPixelsize parameter. By default this is the same as the input pixel (3.2" for the blue and 6.4" for the red bolometer, respectively). The user can set this parameter on the basis of the raster or dithering pattern and on the scan map speed. In order to map any input pixel into the output map as shown in the bottom-right corner of the figure above, the task calculates the sky coordinates of the pixel corners. If the parameter calibration is set to true (default), the task uses for this purpose the same method used by photAssignRaDec for calculating the sky coordinates of the pixel center, that is by using the distortion calibration files (see the description of that task for more detail). If calibration is set to false, than the input pixel is supposed to be a square and the coordinates of the corners are calculated by geometry on the basis of the pixel center coordinates and position angle. The first method is preferable since it takes into account the distortions of the PACS bolometers. The second method is still available for test purposes and for the reduction of the PACS photometer simulator, which assumes squared pixels. Once the corner coordinates are available, first the task transforms the signal from flux(Jy) per input pixel into flux(jy) per output pixel. This is done by dividing the input pixel signal by the area mapped by an input pixel in the output image (the sum of the colored regions in the bottom-right corner of the figure above). The coadded image is obtained with the following method:

$$I_{x'y'} = \frac{\sum_{i=1}^{N_p} a_{xy} w_{xy} i_{xy} / A_{xy,x'y'}}{W_{x'y'}} \qquad (1)$$

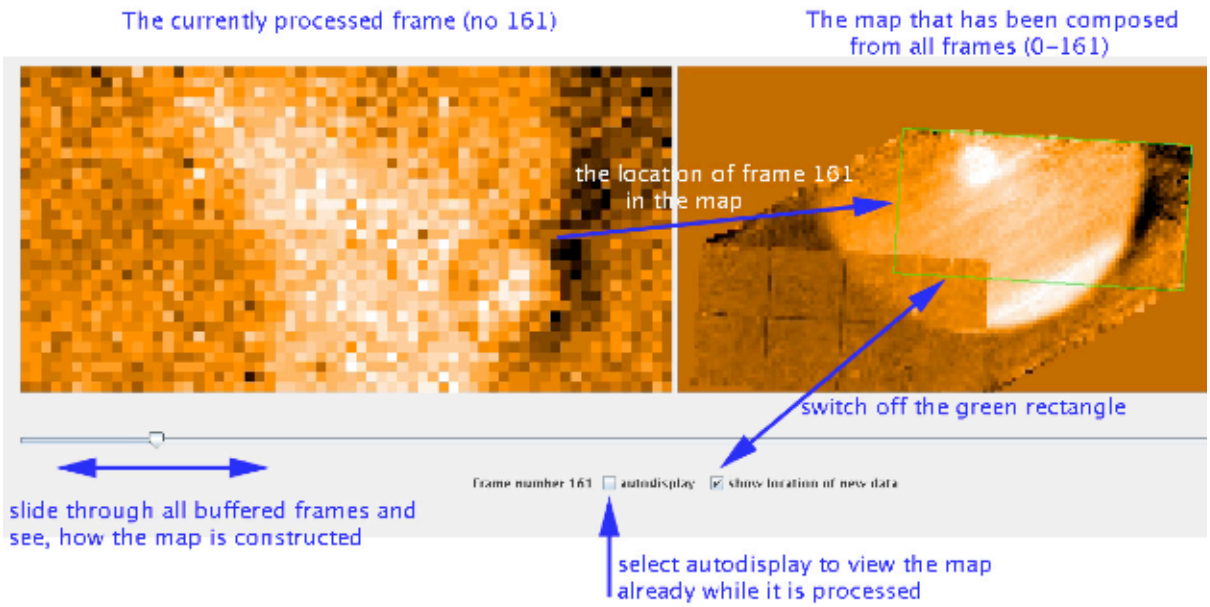
$$W_{x'y'} = \sum_{i=1}^{N_p} a_{xy} w_{xy} \qquad (2)$$

where Np is the number of inoput pixels, I(x'y') is the flux of the output pixel (x',y'), a(xy) is the geometrical

weight of the input pixel (x,y), w(xy) is the initial weight of the input pixel, i(xy) is the flux of the input pixel, A(xy,x'y') is area mapped by an input pixel in the output image (flux conservation factor) and W(x'y') is the weight of the output pixel (x'y'), that is the output coverage map. The geometrical weight a(xy) is given by the fraction of ouptput pixel area overlapped by the mapped input pixel (the 4 regions with different colors shown in the bottom-right corner of the figure above), so 0 < a(xy) < 1. The initial weight w(xy) depends on the observation. In case of chooped-nodded observations (point-source, small-source and raster mode), w(xy) should be given by the coverage map which takes into account the different number of readouts used, pixel by pixel, in the previous averaging processes (averaging of the chopper plateau, averaging of differential on-off images, etc). In the case of scan map observations, w(xy) is just equal to 0 if a pixel is masked out in the available masks (BADPIXEL, SATURATION, GLITCH) and 1 in th opposite case. Thus, the signal Ix'y' of the output image at pixel (x',y') is given by the sum of all input pixels with non zero geometrical (a(xy)) and initial weight (w(xy)), divided by the coverage map (Wx'y', sum of the weights of all contributing pixels, eq. 2).

The task provides as output the final map, the coverage map and the noise map. Only the final map has a correct wcs, the other images are not provided yet with WCS.

10.2.3 Features of the Map Monitor


The currently processed frame (no 161)
The map that has been composed from all frames (0-161)
the location of frame 161 in the map
switch off the green rectangle
slide through all buffered frames and see, how the map is constructed
select autodisplay to view the map already while it is processed

The Map Monitor appears, if PhotProject is started with the option "monitor=1"

The use of the Map Monitor is straight forward. After PhotProject is started with the option monitor=1, the Map Monitor appears and shows how the map is constructed. It has a buffer for all processed frames and maps. The slider moves through this buffer and displays the map in all stages of construction. Here are some remarks:

- autodisplay: if this is selected, the map is immediately displayed, while PhotProject processes the data. Uncheck this option and the buffer initially fills much faster.
- memory: depending on the size of the processed Frames class the buffer may use a lot of memory. Start PhotProject with all memory you can afford. If the Map Monitor runs out of memory, it will delete its buffer to avoid out of memory situations and go on showing only the currently processed map. In this low memory mode the slider is disabled (but it still indicates the number of the currently processed frame).

11 Chopped Raster AOR

11.1 Level 0.5 to Level 1

11.1.1 photMakeRasPosCount (jython prototype available)

See description of the same task in the Point-source pipeline

11.1.2 photAvgPlateau (java prototype available)

See description of the same task in the Point-source pipeline

11.1.3 photDiffChop (java prototype available)

See description of the same task in the Point-source pipeline

11.1.4 photDriftCorrection (jython prototype available)

See description of the same task in the Point-source pipeline

11.1.5 photRespFlatFieldCorrection (jython prototype available)

See description of the same task in the Point-source pipeline
NOTE: In the case of chopped raster mode, only the chop cycles and not the nod cycles are defined for any raster position. Usually, the user can specified the raster observations to reproduce also the nod cycle. This makes the observation dependent on the user observation design. Therefore, the data reduction after this step and up to Level 1 of the pipeline can not be generalized.

11.2 Level 1 to Level 2

11.2.1 photAssignRaDec

See description of the same task in the Point-source pipeline

11.2.2 photProject

See description of the same task in the Point-source pipeline

12 Scan Map AOR

12.1 Level 0.5 to Level 1

12.1.1 photDriftCorrection (jython prototype available)

See description of the same task in the Point-source pipeline

12.1.2 photRespFlatFieldCorrection (jython prototype available)

See description of the same task in the Point-source pipeline

12.2 Level 1 to Level 2

12.2.1 photAssignRaDec

See description of the same task given in the Small-source pipeline

12.2.2 The map reconstruction

At this stage of the data reduction the scan map pipeline is divided in two branches: a simple projection given by Photproject and the inversion given by MadMap. The two methods are implemented to satisfy the requirements of different scientific cases. See following subsections for more details.

12.2.2.1 The simple projection

filterSlew

This task is removing from the science data the readouts corresponding to the satellite slewing, e.g. at the beginning of the science block or between different adjacent scan legs. These readouts need to be

discarded in the map reconstruction because they correspond to a satellite acceleration.

```
>> outFrames = filterSlew(inFrames, [,copy=0]  )

outFrames         : Frames : Frames out with one image per one chopper cycle
inFrames          : Frames : Frames in
copy              : int    :
                      0 - return reference
                      1 - return copy
```

The task is just reading the Status table entry "IsSlew" (see description of "photAddInstantPointing" task for more details). This flag is set to "true" for readouts corresponding to the satellite slewing and "false" elsewhere. The readouts with IsSlew=true are removed from the frames class.

photHighPassfilter (jython prototype)

This task is only a prototype. The purpose is to remove the 1/f noise. Several methods are still under investigation. At the moment the task is just using a Median Filter by removing a running median from each readout. The filter box size can be set by the user (filterbox parameter in the scheme below). By default is is 200 readouts.

```
>> outFrames = photHighPassfilter(inFrames, [filterbox=filterbox], [,copy=0]  )

outFrames         : Frames : Frames out with one image per one chopper cycle
inFrames          : Frames : Frames in
filterbox=        : int    : median filter box size, by default is 200 readouts
copy              : int    :
                      0 - return reference
                      1 - return copy
```

A real high pass filter is still under implementation and its use is under investigation.

photProjects

See description of the same task given in the Small-source pipeline

12.2.2.2 The MadMap case

makeTodArray

Builds time-ordered data (TOD) stream for input into MADmap and derives meta header information of the output skymap. Input data is assumed to be calibrated and flat-fielded. Also prepares the "to's" and "from's" header information for the InvNtt (inverse time-time noise covariance matrix) calibration file.

```
>> PacsTodProduct todProd = makeTodArray(inFrames [,scale=scale] [,crota2=crota2]
                         [,todname=todname] [,toddir=toddir])

inFrames  : Data frames in units of mJy/pixel.   Required input meta-data:
            (1) RA,Dec cubes associated with the frames including the effects of
                distortion.  Assume this step has been previously done by
                PhotAssignRaDec.
            (2) input mask cube which identifies bad pixels.
            (3) information on band (BS,BL,RED), mode (scan/chopped raster),
                and locations between scan legs for data "chunking".

scale     : %pixel scale of output skymap in relation to nominal PACS detector
            size, e.g., 3.2" for Blue and 6.4" Red.
            For scale = 1, the skymap has square pixels equal to nominal
            PACS detector size.

crota2    : CROTA2 of output skymap.  Default = 0.0 degree.

todname   : Filename of TOD file.
```

```
toddir    : Directory that contains the above TOD file.

todProd   : Output product representing the TOD binary bit-stream and associated
            meta data keywords.


Body of todProd is TOD bit stream binary data file consisting of binary header
information and TOD data (Reference:
http://crd.lbl.gov/~cmc/MADmap/doc/man/MADmap.html).
The binary header is four 8byte integers

(1) First sample index for TOD data, set to 0.
(2) Last sample index for TOD data chunk, set to
(n_good_detectors * n_samples) -1.
(3) nnObs = Number of detector values per sky pixel during each time sample
(for default one-to-one mapping of detectors on to sky pixels, nnObs=1).
(4) total number of sky pixels with good data.

The binary header is followed by the data in the order of:

    For each input GOOD detector pixel ("observation"):
           value (double, 8-byte float) ==v
           For each sky pixel observed:
                 weight (4-byte float)  == w
                 skypixel index (4-byte int) == p

    (e.g.,.) for good detectors ii=1,nd and time samples kk=1,nt,
    TOD order is given by:
    for ii=1,nd
          for kk=1,nt
               v[ii,kk]
               for jj=1,nnObs
                  w[ii,kk]
                  p[ii,kk]

Initially for the SPG, we will set nnOBS=1, i.e., use the default one-to-one mapping
of input detectors onto sky pixels.
```

The TOD binary data file is built with format given above and the tod product includes and the astrometry of output map using meta data keywords: CRVAL1 : RA Reference position of skymap CRVAL2 : Dec Referenceposition of skymap EQUINOX : 2000. CTYPE1 : RA---TAN CTYPE2 : DEC--TAN CRPIX1 : Pixel x value corresponding to CRVAL1 CRPIX2 : Pixel y value corresponding to CRVAL2 CDELT1 : pixel scale of sky map (=input as default, user parameter) CDELT2 : pixel scale of sky map (=input as default, user parameter) CROTA2 : PA of image N-axis (=0 as default, user parameter) The weights are set to 0 for bad data as flagged in the mask. Dead/bad detectors (detectors which are always {or usually} bad), are not included in TOD calculations. The skypix indices are derived from the projection of each input pixel onto the output sky grid. The skypix indices are increasing integers representing the location in the sky map with good data. The skypixel indices of the output map must have some data with non-zero weights,must be continuous, must start with 0, and must be sorted with 0 first and the largest index last. Future planned parameters that may be implemented include: medianSub : True/False; Flag to subtract median value from input data (default = false). nnObs : Number of detector values per sky pixel during each time sample (for default one-to-one mapping of detectors on to sky pixels this value is one (i.e., the value for each sky pixel for one time sample is based on only one detector value). If value for a sky pixel for one time sample is based on multiple values, then nnObs > 1 and one needs to assign the appropriate weights (e.g., fractional area of detector pixel seen by sky pixel, and conserve surface brightness. maxGap : Maximum size of gap (in samples) before chunking is done. otfName : On-target-flag name (ONTARGET, HSC-DOC-0662 [PACS-PTREQ-G08] which will be a status flag from the pointing product. Required for data chunking of scan data by scan leg.

runMadMap

The module runMadMap is the wrapper that runs the JAVA MADmap module. MADmap uses a maximum-likelihood technique to build a map from an input Time Order Data (TOD) set by solving a system of linear equations. It is used to remove low-frequency drift ("1/f") noise from bolometer data while preserving the sky signal on large spatial scales. (Reference: http://crd.lbl.gov/~cmc/MADmap/doc/man/MADmap.html). The input TOD data is assumed to be calibrated and flat-fielded and input InvNtt noise calibration file is from calibration tree.

```
>> SimpleImage map = runMadMap(todProd, [calTree=calTree] [,filterLength=filterLength]
                    [,maxRelError=maxRelError] [,maxIterarions=maxIterations] )

todProd       : The PacsTodProduct from makeTodArray

calTree       : PacsCalibrationTree containing calibration InvNtt information stored
                as an array of size max(n_correlation+1) x n_all_detectors.
                Each row represents the InvNtt information for each detector.

filterLength  : Specifies the length of the FFT's that will be done; code will make
                a best guess if not provided.

maxRelError   : Maximum relative error allowed in PCG routine (default is 1e-6).

maxIterations : Maximum number of iterations in PCG routine  (default is 50).


map           : Simple image sky map including header information from the tod
                product meta-data.  Products also include the naive map
                (map without corrections), a coverage map, and a representative
                noise map (product definition is TBD).
```

The filterLength, which is calculated by the module, must be larger than 2*bandWidth, and can be much longer. For optimum performance filterLength should be the smallest power of two such that filterLength / (ln(filterLength) + 1) >= bandWidth - 1. But, note that for best performance filterLength should not be longer than the stationary time scale of the the noise. Future planned parameters that may be implemented include: bandWidth : Width of the non-zero band along the diagonal of inv(N); code will derive from noise file if not provided. The bandWidth is 2*n_correlation +1. maxMemory : Maximum number of bytes of memory that each process can allocate (default is 1GB). medianSub : Flag to subtract median of the input data values before MADmap computation, and then the median level is added back into the output sky map. May be helpful for data of limited dynamic range where background >> signal.

13 Trend Analysis Product generation

This section is dedicated to the trend analysis product generation. The concept and the scheme of this product generation has to be still finalized. At the moment only the calibration blocks and several HK of each observation are saved as trend analysis products. The tasks responsible for these products are listed and described below. However, it is worth to mention that the implementation of these tasks and their results is prone to change on the basis of calibration scientist requirements.

13.1 photTrendCS

This task is not mandatory for the pipeline. For trend analysis purpose, photTrendCS collects, reduces and stores useful data about the internal calibration sources. This process is applied for each calibration block encountered. Facultative, this task leaves the frames unchanged.

```
>> outFrames = photTrenCSTask(Frames inframes [,newVersion=newVersion][,calTree=calTree]
               [,context=context][,copy=copy])

outFrames    : Frames              : Frames isn't changed
inFrames     : Frames              : Frames with signal in Volt
newVersion   : Boolean             : forced the creation of a new version
calTree      : PacsCalibrationTree : calibration tree containing all calibration products
               used by the pipeline
```

```
context      : QualityContext      : quality context collected at the end of the pipeline
copy         : int                 :
                                   0 - return reference (default)
                                   1 - return copy
```

Content of CS product generated : CS product contains three parts {red, green, blue}. While the red part is always filled, blue and green depends on the current observation. A keyword called 'channel' and stored in the metadata, keeps information on the valid filling part (green or blue). Here are the keywords and information stored:

- "CS1" in volt, contains CS1 data stored in one calibration block.
- "CS2" in volt, contains CS2 data stored in one calibration block.
- "CS1-Time" gives the time in microseconds (since 01Jan1958) of each layer of CS1 data cube.
- "CS2-Time" gives the time in microseconds (since 01Jan1958) of each layer of CS2 data cube .
- Meta data
  - "Channel" = { green, blue} gives information on the valid part.
  - "CS1-CPR" contains the mean of the chopper positions extracted during an observation of the CS1 - Value is in command unit (CU)
  - "CS2-CPR" contains the mean of the chopper positions extracted during an observation of the CS2 - Value is in command unit (CU)
  - "CS1-Temperature" is the mean of the temperature in Kelvin for the first calibration source
  - "CS2-Temperature" is the mean of the temperature in Kelvin for the second calibration source
  - "CS1-SigmaTemp" is the standard deviation in Kevin of the temperature of CS1
  - "CS2-SigmaTemp" is the standard dev iation in Kelvin o fthe temperature of CS2
  - "Bias" contains the mean of each BU (Buffer Unit) of the difference between VH and VL. This quantity is in Volt
  - "Mode" gives the reading mode led by the warm electronic BOLC This quantity is a string = {Direct,DDCS} takes from the median of the calibration block.
  - "Gain" of the warm electronic, possible values are { 0 = high gain, 1= low gain } . This value is based on the median of the value found into the calibration block.

13.2 photDiffCStoring

Not mandatory, this task reuses table HkCalBlockTable stored in the frame in order to generate dCS data on disk. Once these data are stored on disk, elaborated algorithms can be applies by the task photDriftCorrection. Otherwise, only dara stored in the memory will be used. Each calibration block encountered generates one dCS file in the pool.

```
>> outFrames = photDiffCStoring(inFrames [,calTree=calTree][,newVersion=newVersion]
              [,quality=quality][,copy=copy])

outFrames : Frames              : the returned Frame object
inFrames  : Frames              : the input frame object containing the difference and
                                  the average of the images per plateau.
calTree   : PacsCalibrationTree : calibration tree where each leaf contains a calibration
                                  product
newVersion: Boolean             : forces the creation of a new dCS version in the
                                pool storage
                               * false (jython: 0) - no new version is created
                               * true  (jython: 1) - new version is created
copy      : boolean with the possible values :
                               * false (jython: 0) - inFrames will be modified and returne
                               * true (jython: 1)  - a copy of inFrames will be returned
quality   : QualityContext      : the quality context
```

Each dCS information stored contained the following information

- channel = {"red","green","blue"} depends on the channel currently processed
- startTime is the start time of the calibration block

- endTime is the end time of the calibration block
- meanTime is the middle time of startTime and endTime - this time is used during the research
- cs1Temperature is the temperature of the first calibration source in kelvin
- cs2Temperature is the temperature of the first calibration source in kelvin
- CS1-Sigma is the standard deviation of the temperature of the first calibration source
- CS2-Sigma is the standard deviation of the temperature of the second calibration source
- CS1-CPR is the chopper position when the instrument looks at the first calibration source
- CS2-CPR is the chopper position when the instrument looks at the second calibration source
- Mode = { "Direct","DDCS" } respectively direct readout or Double Differential Correlated Sampling readout
- Gain = {0,1} so respectively high or low gain
- Bias in volt is the mean of VH-VL voltage of all BU (Buffer Unit)
- Frequency of the chopper between CS1 and CS2 in Hz
- filename is a unique name - useful if we want to regenerate this file into an other format (FITs)

14 Raw Telemetry to Level 0

Usually the pipeline data reduction is supposed to start directly from the Level 0 products. However, these tasks can still be usefull for test purposes.

14.1 averageFrames

Average Photometer detector signals (avgNr readouts) for the raw data instrument modes .

```
>> Frames outFrames = averageFrames(Frames inFrames, int avgNr)

outFrames : Frames : Frames out
inFrames  : Frames : Frames in
avgNr     : int    : Number of samples to average
copy      : int    :
                     0 - return reference
                     1 - return copy
```

- Frame Signals (double) are just averaged
- Frame Wavelengths (double) are just averaged
- The Frame Masks (boolean) are reduced. Optional user may reduce this data by "AND" or "OR" operations against True.
- The Frame Status (int) are averaged. But the then rounded to the nearest Integer.
- The Frame Status (boolean) are treated as Frame Masks
- The Frame Status (string) are selected by majority, in case of no majority the first one is taken

14.2 readTm - reading Raw Telemetry

Reading raw telemetry from a PacketRecorder archive file (.tm file) .

```
>> seq = readtm()

seq : PacketSequence : PacketSequence containing raw Tm and/or TC SourcePackets
```

In the operational environment this steps will be hidden for the general user. But, of course, within interactive sessions it ois possible to execute every single step and examine the intermediate results. This will open a file selector box showing all files in your working directory that end with "`.tm`". The telemetry is then loaded from the selected file into a PacketSequence variable called `seq`. If the `pacs.tm.datapath` property is set to an existing directory, the file selector box will be opened in that directory which makes it easier to navigate from there to your data. The`readtm()` also accepts a filename in which case no file selector box will pop up.

14.3 extractDataframes - decompress the science tm packets

This step generate the intermediate Product Decompressed Science data.

```
>> dfs = extractDataframes(seq)

seq : PacketSequence    : PacketSequence containing raw Tm and/or TC SourcePackets
dfs : DataframeSequence : Sequence containing the raw, decompressed DataFrames
```

Again this step is hidden in the pipeline Level 0 data generation. User may use it as long as the pipeline Level 0 generation is not available or for debugging purposes. The `extractDataframes` task groups the science telemetry packets per group that can be decompressed together and decompresses them. The result is a DataFrameSequence, a collection of PacsDataFrame objects. These are decompressed buffers of the two Signal Processing Units (SPU).

14.4 decomposeDataframes – organize the raw decompressed data in `Frames` and `PhotRaw` data structures

`Decompose the raw DataFrames into Products suitable for further processing.`

```
>> pacsMix = decomposeDataframes(dfs [,channel=channel] [,mode=mode]
             [, calVersion = calVersion])

pacsMix : PacsMix           : Container for the Products (Frames, Ramps, PhotRaw)
dfs     : DataframeSequence : Sequence of Pacs Dataframes
channel : String  :
                    "red"  - red channel only
                    "blue" - blue channel only
                    "both" - both channel (default)
mode    : String  : default is all modes
                    "frames"   - only frames
                    "ramps"    - only Ramps
                    "subramps" - only Subramps
                    "rawramps" - only Raw Ramps
calVersion : String : Version of the calibration files used
```

`PacsDataFrames` contain the result of the Decompression (reduced data, raw data, DecMec data and Compression Header data). This pipeline step is restructuring the data in a proper format for further scientific analysis. This proper formatted products are`Frames` and `PhotRaw`, depending on the instrument algorithm and compression mode. `Frames` contain the reduced data as data cube, collapsed DecMec information, and decoded Label information in the associated Status. The `PhotRaw` product contain raw channel data (non averaged data) e.g. of the rotating additional raw channels.`Pixel deselected with the Detector Selection Table are masked by the BLINDPIXEL mask.decomposeDataframes` return a `PacsMix` . Depending on the (possibly different) :

- Instrument Algorithms
- Compression Modes
- User selections (Red channel and / or blue channel)

the user the `PacsMix` contain one or more`Frames` and/or `PhotRaw` products. For`Frames` the DecMec data are collapsed from the full readout sampling to the frequency of the reduced data :

- OBSID : first entry of the associated block of DecMec data
- BBID : first entry of the associated block of DecMec data
- LBL : first value + decoded value + check whether it change
- TMP1 : first value
- TMP2 : first value
- FINETIME : first value
- VLD : first value + check it is not changing
- CPR : mean value
- WPR : mean value
- BOLST : mean value

- CRDC : first value :
- CRDCCP : first value
- DBID : first value
- BSID : first value

Calibration Files :

- `FilterBandConversion` : Filter to Band Conversion.
- `LabelDescription` : label description.

Going from Level 0 to Level 0.5 implies extracting/collecting the necessary auxiliary data.

14.5 readAttitudeHistory

Read the attitude history.

```
>> attitude = readAttitudeHistory(pdfs)

attitude :
pdfs     : DataframeSequence : Sequence of Dataframes
```

Reads the instantaneous pointing product covering the same time as the dataframes in the DataframeSequence pdfs.

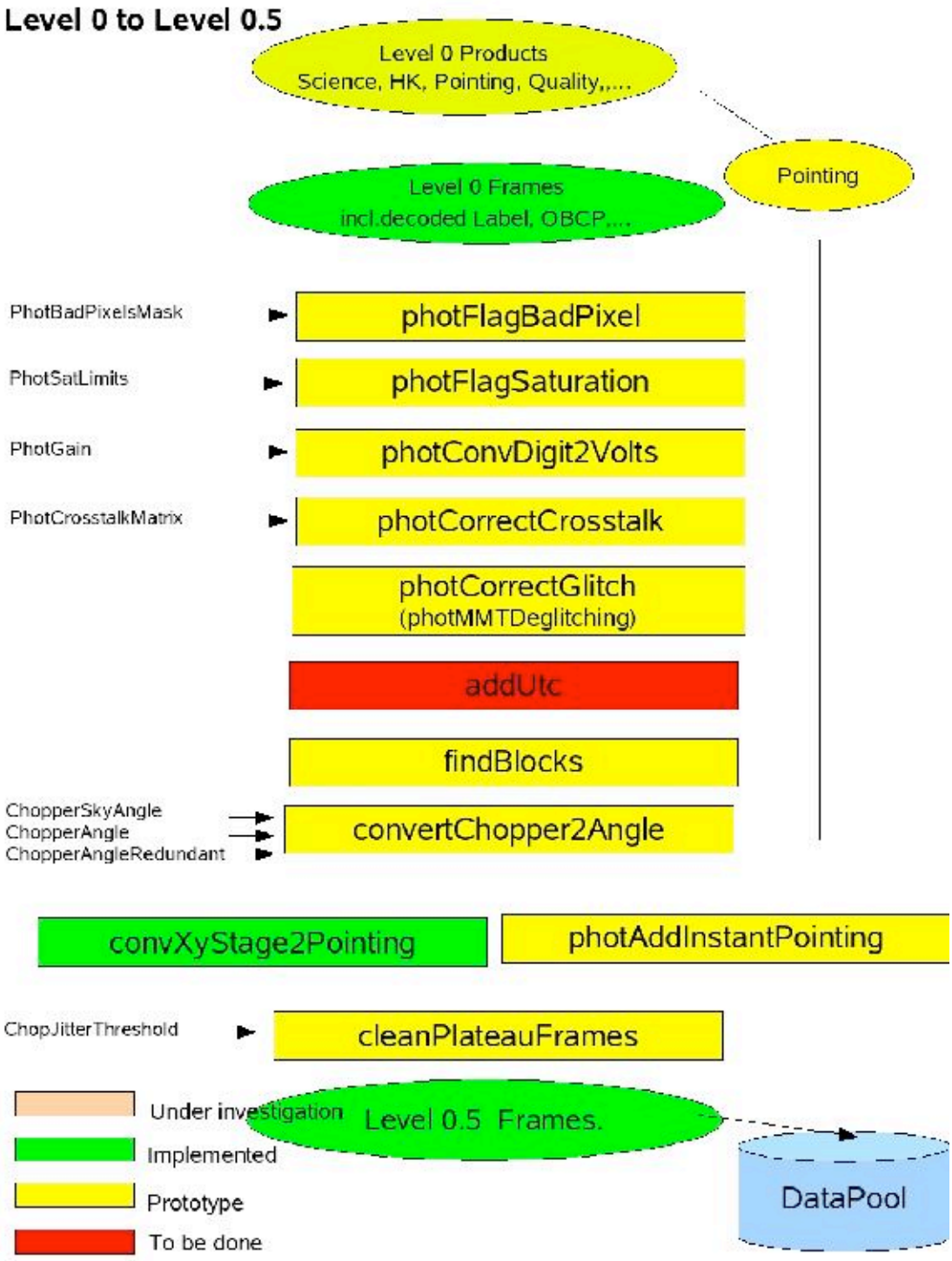
14.6 readTimeCorellation

Read the Time Correlation information.

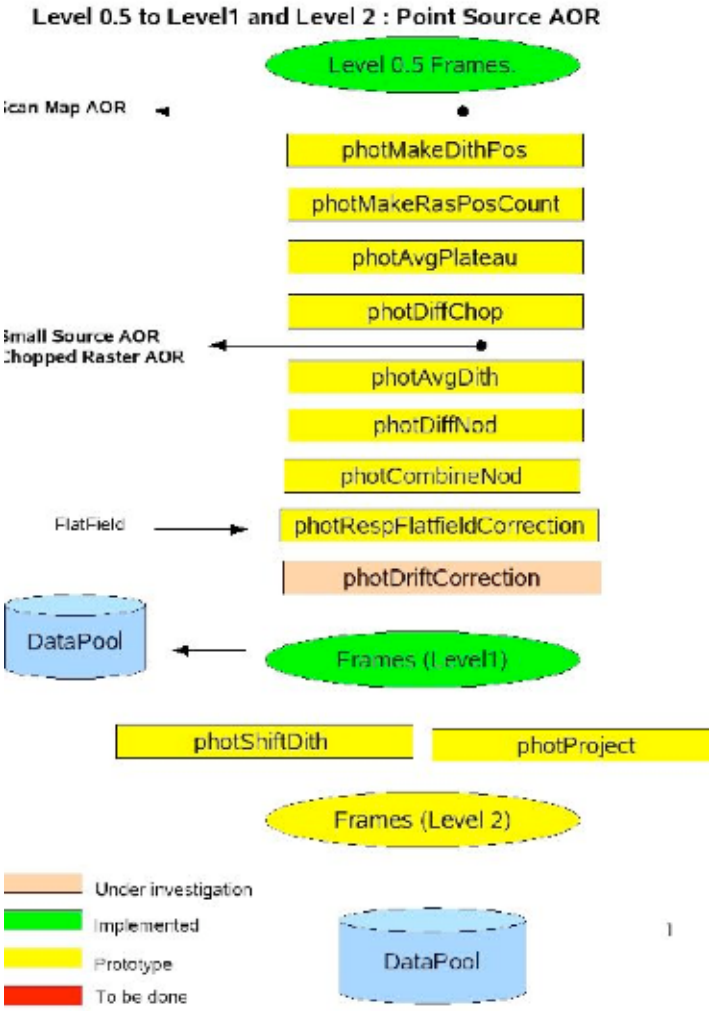
```
>> timecor = readTimeCorellation(pdfs)

timecor : TableDataset : Time correction values
pdfs    : DataframeSequence : Sequence of Dataframes
```

Reads the time corellation product covering the same time as the dataframes in the DataframeSequence pdfs.

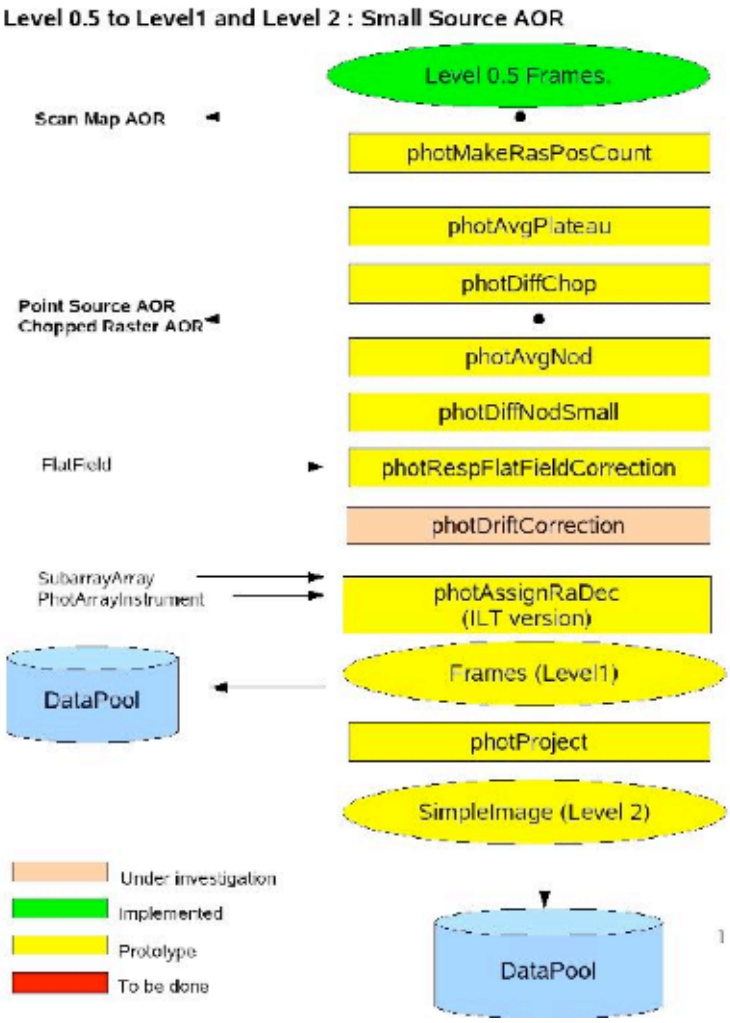
15 SPG Pipeline chart



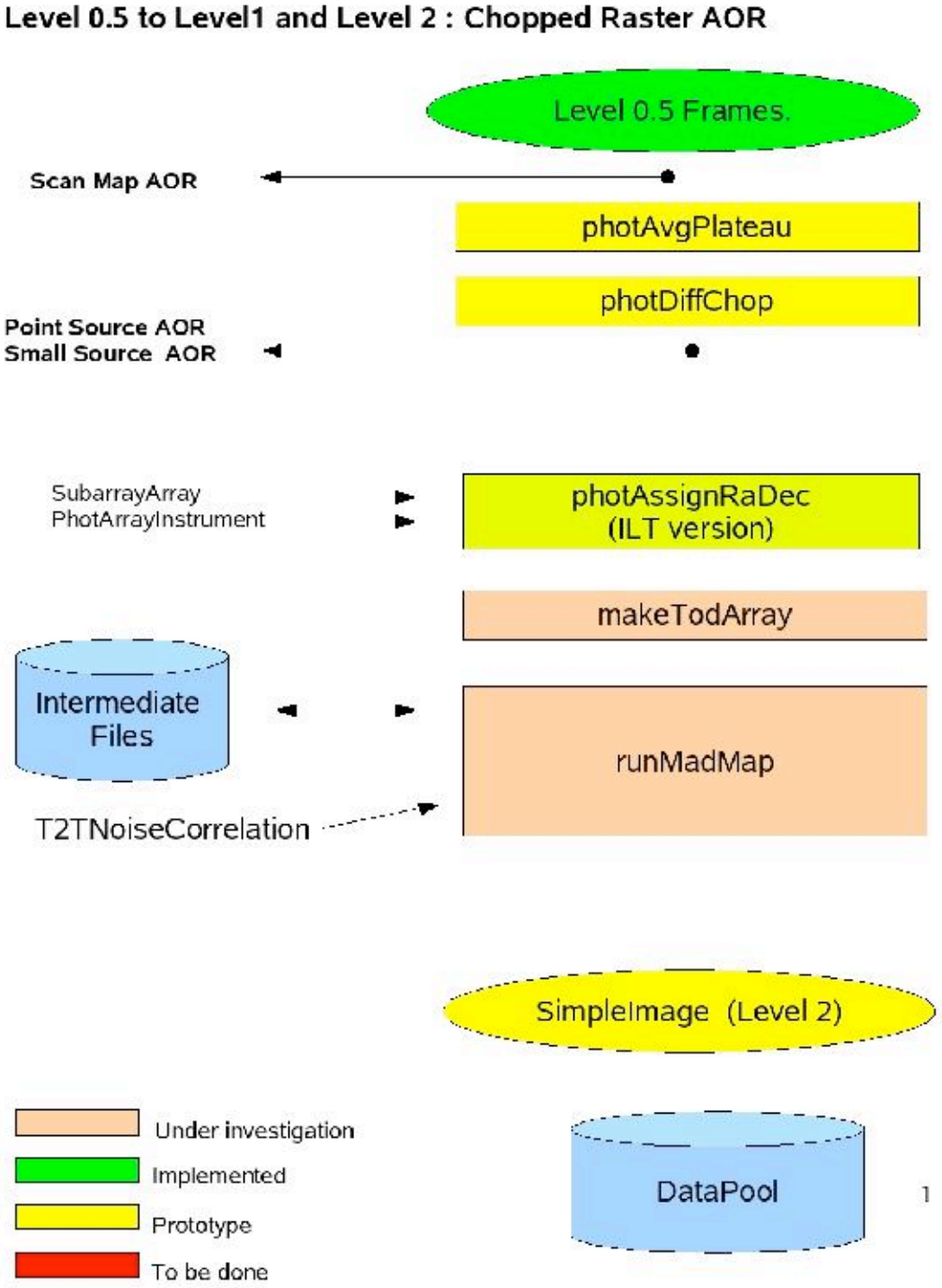
SPG pipeline chart level 0 to 0.5



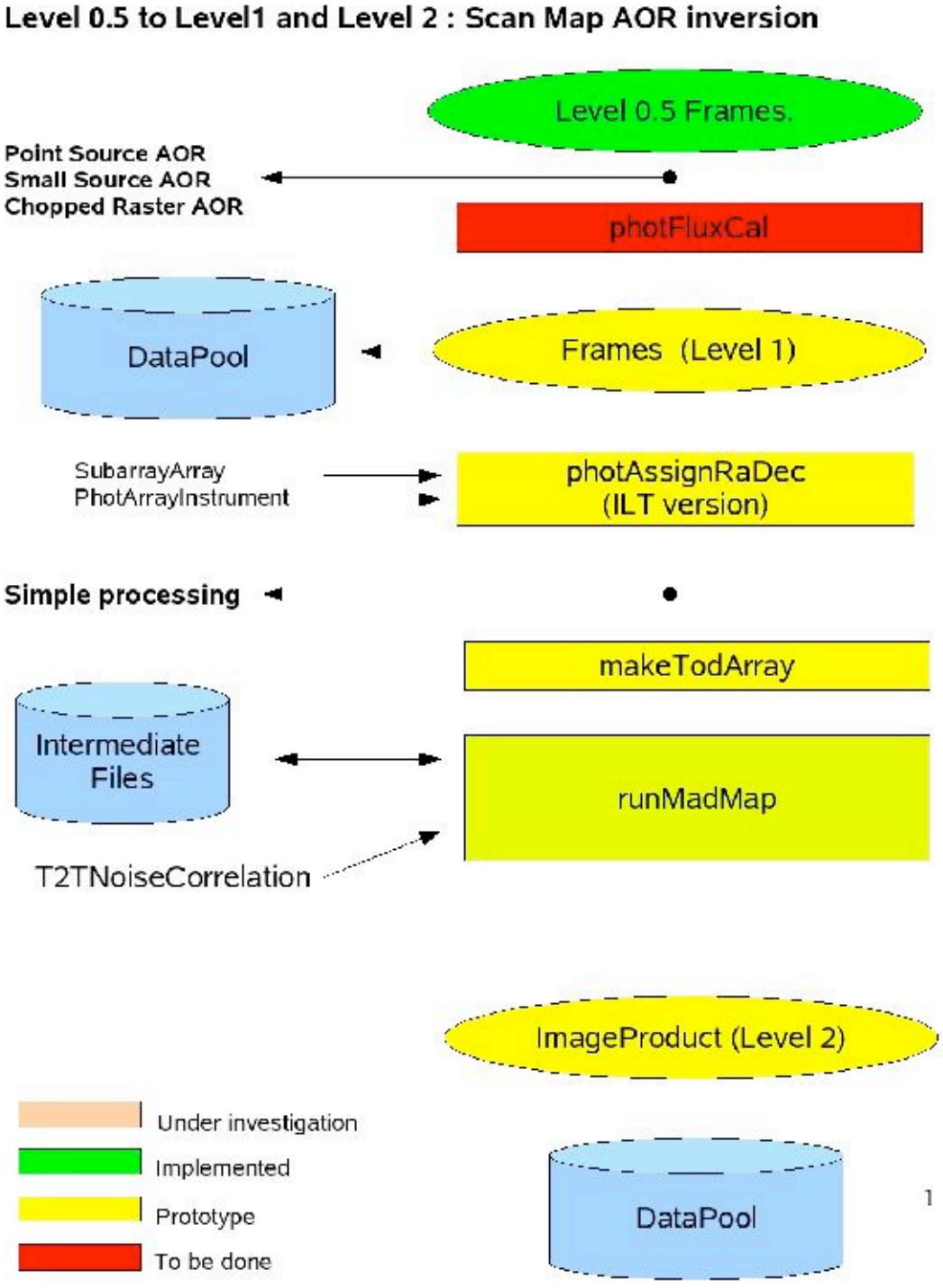
SPG pipeline chart level 1 to level 2 : Point Source AOR



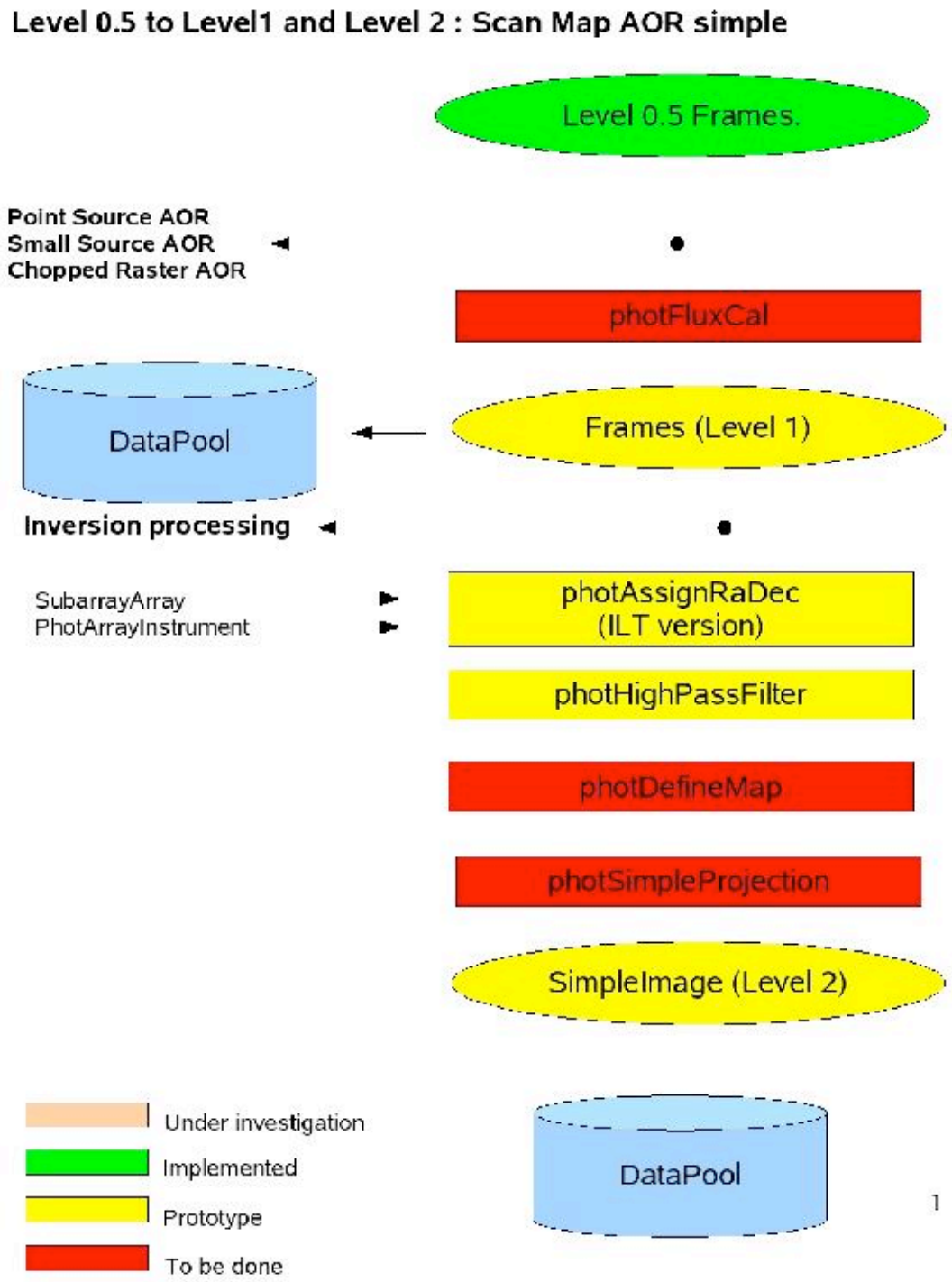
SPG pipeline chart level 1 to level 2 : Small Source AOR



SPG pipeline chart level 0 to level 1 : Chopped Raster AOR



SPG pipeline chart level 1 to evel 2 : Scan Map AOR (inversion)



SPG pipeline chart level 1 to level 2 : Scan Map AOR (simple)

16 Product summary

Overview - last updated 2006/06/09

Level	Product name	Status
0	readTm	Done
0	extractDataframes	Done

0	decomposeDataframes	Done
0.5	readAttitudeHistory	toDo
0.5	readTimeCorrelation	toDo
0.5	extractDMC	toDo
0.5	extractFrames	Done
0.5	photFlagSaturation	Done
0.5	photConvDigit2Volts	Done
0.5	photFlagBadPixels	Done
0.5	photMMTDeglitching	Done
0.5 - 1	decodeLabel	Done
0.5 - 1	findBlocks	Done
1	convChopper2Angle	Done
1	convXYStage2Pointing	Done
1	photAddInstantPointing	Done
1	photCorGlitch	Done
1	cleanPlateau	Done
1	photAvgPlateau	Done
1	photAssignRaDec	Done
1	photDiffChop	Done
1	photAvgNode	done
1	photDriftCorrection	to be tested
1	photRespFlatfieldCorrection	to be tested
1	cal-block pre-processing	to be finished
1	addUTC	toDo
1	photProject	done
1	phoHighPassFilter	to be further developed

17 Appendix

17.1 How to remove sky background and telescope emission

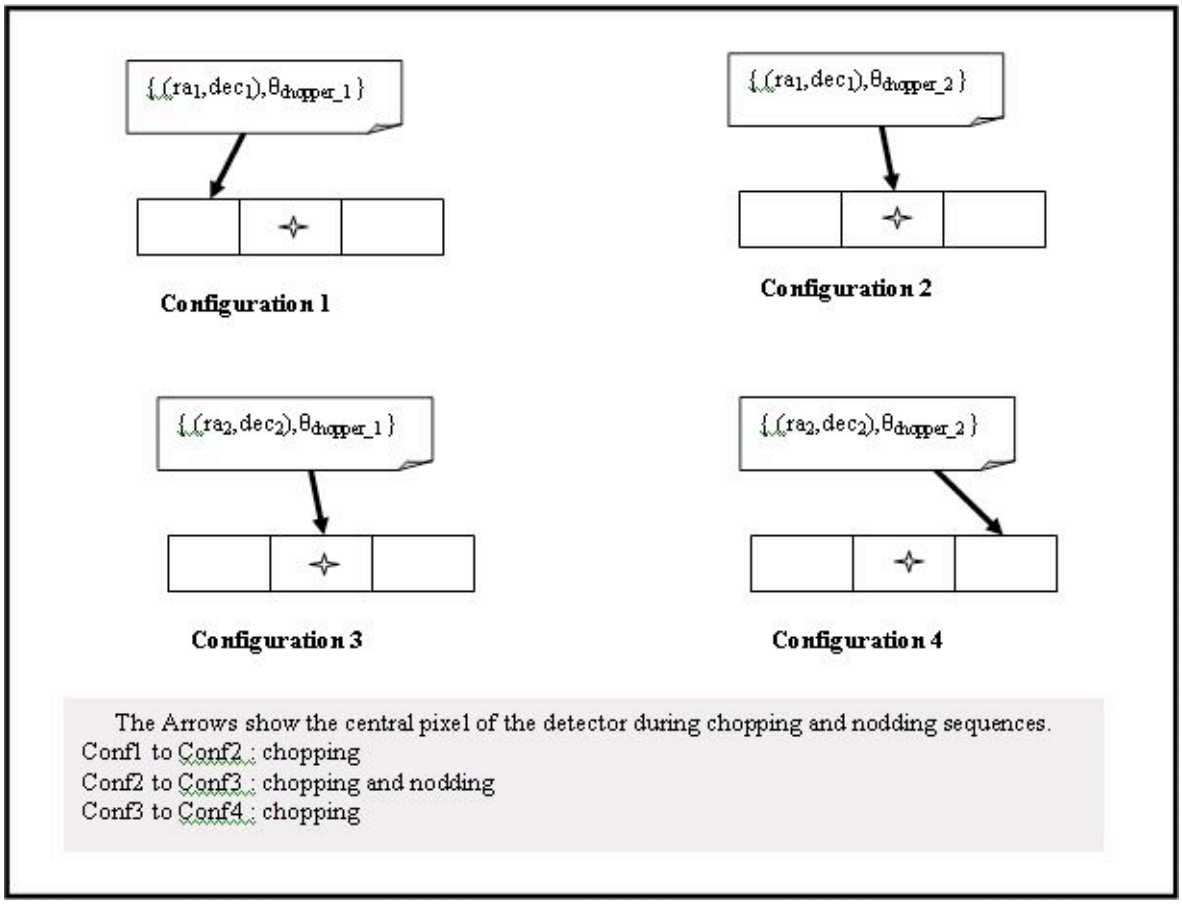
Telescope emission is the major flux received by the detector. During his life telescope temperature should be to 80Â°K on average and his emissivity to 4%. In order to remove instrumental error, chopping and nodding mode are used. If the chopper doesnÂ't move, the optical path in PACS doesnÂ't change when the telescope pointing is running. Successive pointing positions in the sky are lead by the satellite pointing and are called nodes. If the chopper is moving, the optical path in PACS instrument change. So that, the flux received by the detector doesnÂ't have the same telescope emission. Three sky areas are used:

- one containing a brightness source,
- and two others with no source

Four observing configurations exist:

```
Configuration 1 : first position : point-source is observed 1
```

```
Configuration 2 : after chopping                          2
Configuration 3 : after nodding and chopping              3
Configuration 4 : after chopping                          4
```



Principle

Basic equations

s = source
f = foreground (telescope emission)
b = background

1. Initial configuration, no brightness source exists:
   1 $i_1 = 0 + f_1 + b_1$ (1)
2. Satellite pointing is unchanged; chopping angle is changed in order to have brightness source in the field of view:
   1 $i_2 = s + f_2 + b_2$ (2)
3. Satellite pointing has been changed; chopping angle is changed in order to align the previous source in the same previous matrix area (case 2).
   1 $i_3 = s + f_1 + b_2$ (3)
4. Satellite pointing is unchanged; chopper angle is changed
   1 $i_4 = 0 + f_2 + b_3$ (4)

Now, it is possible to remove the telescope background emission:

- i_2 Â- i_1 = s + (f_2Â-f_1) + (b_2Â-b_1) (5)

- i_3 Â- i_4 = s + (f_1Â-f_2) + (b_2Â-b_3) (6)
- [(i_2Â-i_1)+(i_3Â-i_4)] / 2 = s + b_2 Â- (b_1+b_3)/2 (7) the mean of the two couple of images
- [(i_2Â-i_1)+(i_3Â-i_4)] / 2 = s + (b_2 Â- b_1) (8) if b_1 = b_3
- [(i_2Â-i_1)+(i_3Â-i_4)] / 2 = s (9) if b_1 = b_2 = b_3

If the sky background is different between two positions, it wonÂ't be possible to correct the sky target properly.

Glossary

Stack
: sorted list of the observations of an astronomical object got in a given context. All of these observations can be processed in the same way.

Plateau
: data sequence got during an elapsed time while the chopper and telescope pointing are unchanged.

Cycle
: one ON/OFF chopper sequence

Node
: telescope pointing is fixed (only slight motion due to the jitter can be found)